A MARSHALL CAVENDISH **32** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95



INPUT

Vol. 3 **No 32**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Graeme Harris. Pages 985, 986, 987, 988, Berry Fallon Designs/Projection Audio Visual. Pages 993, 996, Graeme Harris. Pages 998, 999, 1001, 1002, 1003, Kevin O'Keefe. Pages 1010, 1012, 1013, 1014, 1015, Ellis Nadler. Pages 1011, 1012, 1013, 1015, Dave King.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN
Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015
New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington
Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

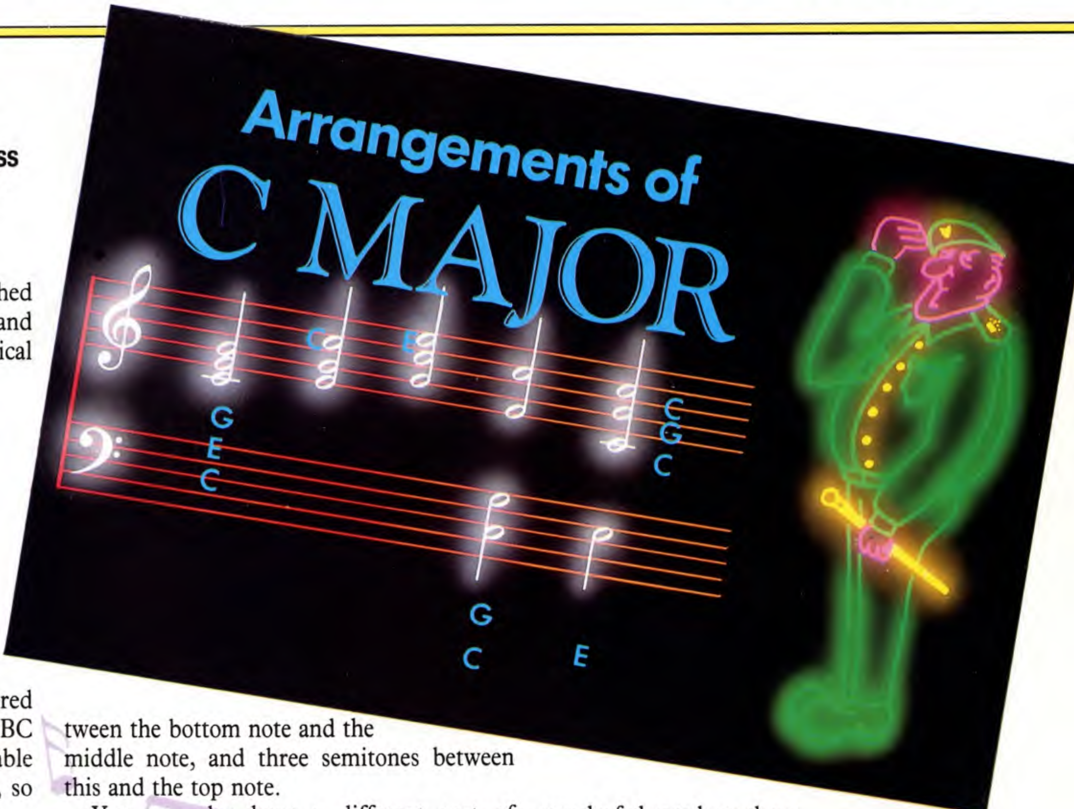
In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and + | COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+ | DRAGON 32 and 64
ZX81 | VIC 20 | TANDY TRS80 COLOUR COMPUTER

PLAYING IN HARMONY

- INCORPORATING MUSIC DATA
- CHANGING THE TEMPO
- COMPLETING THE HARMONY
- GENERATING A NEW HARMONIZING CHORD

There's no need to stick to one-finger tunes, with our Playing in Harmony program you can progress to playing chords or simultaneous melodies and harmonies



The two previous musical articles published in *INPUT* discussed simple music theory and how to turn your computer into a musical instrument (pages 669–675), then how to incorporate music data in a program which subsequently scans through and plays the sorted tune (pages 701–707). But both these concentrated only on a single melody line.

This article goes one step further and provides programs that allow chords, or simultaneous melodies and harmonies to be played. Although the theory of this has a general application, of the computers covered here, only the Commodore 64 and the BBC have sound chips which render them capable of playing more than one note at a time, so only programs for these two are included.

The programs given here adopt the same conventions as the two previous articles, so you may want to reread these and refresh your memory before you tackle the new techniques.

WHAT IS A CHORD?

A chord is simply any group of notes played simultaneously. If you play a handful of random keys on a piano, that produces a chord, although—confusingly—the sound produced may be discordant. The commonest and most pleasing chords contain three notes, and are related to the major **do, re, mi** scale discussed previously.

If you take **do**, the note two notes above it, **mi**, and the note two notes above that, **so**—these are C, E and G in the scale of C major—and play these notes together, this is an example of the simplest and best known of all chords—the major chord. The name of the chord is derived from the note it is built up on: if this note is C, it is called the chord of C major.

Any major chord has four semitones between the bottom note and the middle note, and three semitones between this and the top note.

You can also have a different sort of chord—a minor chord—which has three semitones between its bottom note and its middle note, and four between the middle note and the top note. So major and minor chords both span seven semitones, but their middle note is slightly higher or lower. This gives the two chords different sound quality: the major chord is usually considered brighter, and the minor chord sadder.

You can build up one other type of chord in this way in a major scale. Starting on **ti**, or B in the scale of C major, this chord contains B, with D and F from the next octave up. This has three semitones from its bottom to its middle note and three from its middle to top note. This is called a 'diminished' chord, and is used much less frequently than the major and minor chords.
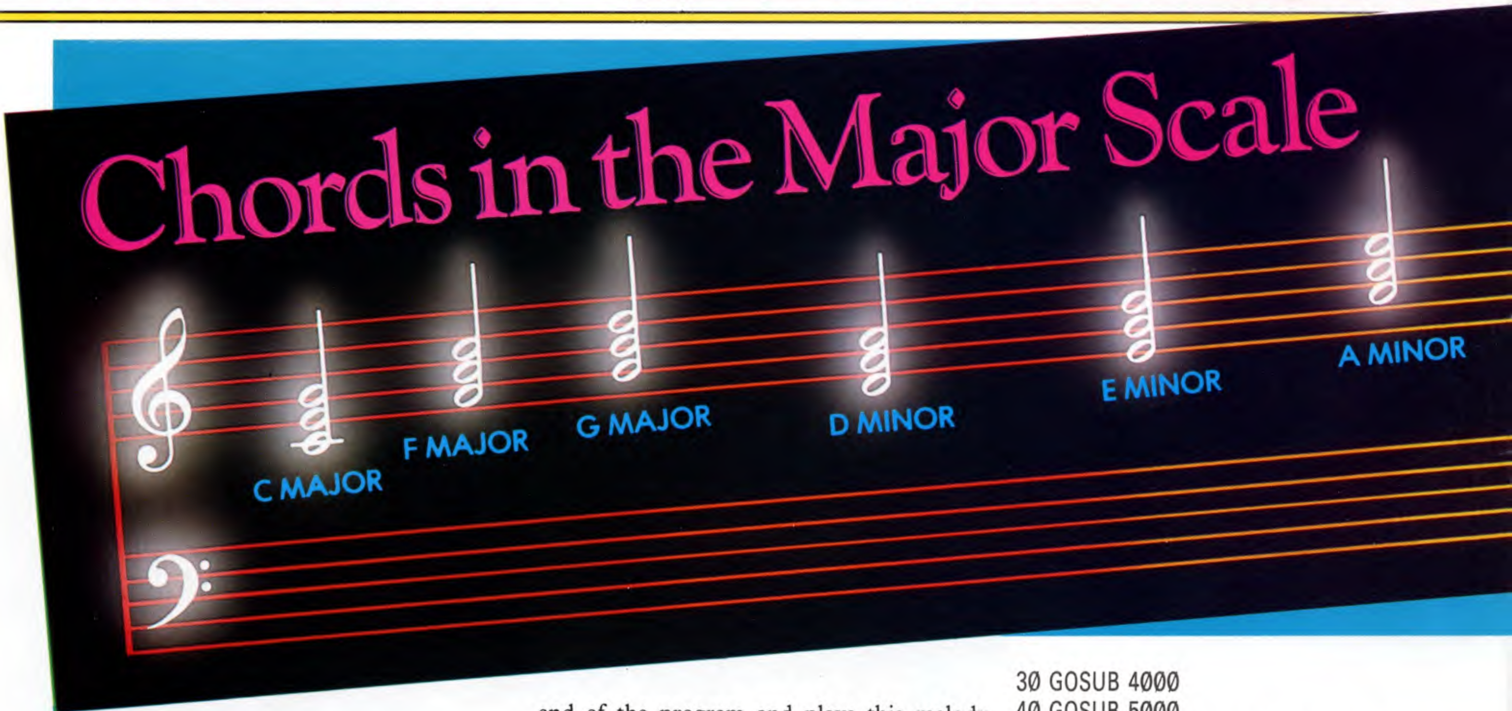
There are seven notes in the major scale, on which can be built three major, three minor and one diminished chord. The three types of chord are sometimes known as triads, since they have three notes, and the note on which the chords are built is known as the 'root', so the root of E minor is the note E. The diagram overleaf shows how these chords might appear in sheet music. Notes that are to be played simultaneously appear one above the other.

HOW CHORDS ARE USED

If a tune in C major has the note C in it at a given point, then any chord that also contains C will harmonize with it and produce a pleasing sound, richer than the melody on its own.

The major chords C and F major and the minor chord A, all contain the note C, so all three of these chords will harmonize with a melody note C. The melody note can act as either the top, middle or bottom note of a triad, so any note can be harmonized with three different triads. C, for example, is the bottom note of C major, the middle note of A minor and the top note of F major. In fact, if a melody contains C, then only an additional two notes are needed to give the complete three-note chord; E and G, to give the chord C major. In this case the melody note functions both as part of the melody and also as part of the chord.

Chords in the Major Scale



In the chord of C major, C need not always be the lowest note. It doesn't matter which note is at the top or bottom, or even if there is more than one of any of the constituent notes. Arrangements which don't contain C at the bottom of the chord are known as 'inversions' of the chord. The diagram on the previous page shows a few different arrangements of notes, all of which make up a chord C major. The same principle applies, of course to other chords.

HARMONIZING

The following program reads in a single melody line from the DATA statement at the end of the program and plays this melody along with two notes, a little lower in pitch than the melody, which harmonize with it.

A random process is used to generate the other two notes of the three-note chord, so the actual harmony will vary each time the tune is played, but the chord produced is always a triad from the key of C major. As it stands, the DATA statements already contain a melody, but you could try any tune in this program, so long as it is in C major: in other words make sure that the **do** in your tune corresponds to C, with pitch number 13. Also don't let the melody fall below this note. This still gives you two octaves to play with, and the lowest octave, with pitch numbers below 13, will contain the two notes completing the harmonization.

The notes in all three voices change at the same time. Whenever the melody note changes, a new harmonizing chord is generated. The program will produce sensible harmonies, though not brilliant ones.

The melody 'When the Saints Go Marching In' is stored in the DATA statements at the end of the program; one bar of music is stored in each DATA statement. The diagram (right) shows the opening of two variations that the program may produce. The melody is the same in each case, of course, but the harmony varies. With the first one, the four chords are F major, C major, B diminished and G major, and the second is C major, E minor, D minor and C major.

```
10 GOSUB 3000
20 INPUT "♥TEMPO";TP
30 GOSUB 4000
40 GOSUB 5000
90 K1=1:K2=2:K3=3:K4=4:K5=5
100 READPV,T:IFPV=0THEN180
110 C=TB(PV)
120 I1%=RND(K1)*K2+K2:IFI1%=K3THEN
   I2%=K5:GOTO135
130 I2%=RND(K1)*K2+K4
135 I1%=C-I1%:I2%=C-I2%
140 POKEL1,LQ%(PV):POKEH1,
   HQ%(PV)
150 POKEL2,LQ%(TA(I1%)):
   POKEH2%,HQ%(TA(I1%))
160 POKEL3,LQ%(TA(I2%)):
   POKEH3,HQ%(TA(I2%))
170 POKEE1,EN:POKEE2,EN:
   POKEE3,EN
180 FOR DL=1TOTP*T:NEXT
190 POKEE1,EF:POKEE2,EF:
   POKEE3,EF
200 GOTO100
3000 SI=54272
3010 EN=33: EF=32
3020 FOR I=SI TO SI+28: POKE I,0: NEXT
3030 POKE SI+5,16*1+8
3040 POKE SI+6,16*15+8
3050 POKE SI+12,16*1+8
3060 POKE SI+13,16*11+8
3070 POKE SI+19,16*1+8
3080 POKE SI+20,16*11+8
3100 POKE SI+24,6
3110 L1=SI: H1=SI+1:E1=SI+4
3120 L2=SI+7:H2=L2+1:
   E2=SI+11
3130 L3=SI+14:H3=L3+1:
   E3=SI+18
3140 RETURN
4000 DIM HQ%(37), LQ%(37)
4010 TMP=2227:P2=2↑(1/12)
```

Microtip

Softer Sound on the Commodore

In the first program in the main article the SID chip is programmed to produce very characteristic bold sounds using saw-tooth waveforms.

The chip can produce different sounds using different waveforms by POKEing with different numbers. The Line to look at is Line 3010. Originally EN and EF were set to 33 and 32. If you change the values to 17 and 16, you are instructing the SID chip to produce more pleasing sounding triangular waves instead.




```
4Ø2Ø FOR I=1 TO 37
4Ø3Ø LQ%(I)=TMP-256*INT
   (TMP/256): HQ%(I)=TMP/256
4Ø4Ø TMP=TMP*P2
4Ø5Ø NEXT: RETURN
5ØØØ DATA 1,3,5,6,8,1Ø,12,13,15,
   17,18,2Ø,22,24,25,27,29,3Ø,32,
   34,36,37
5Ø1Ø DIM TA(22):FOR I=1 TO 22:
   READ TA(I): NEXT
5Ø2Ø DATA 1,1,2,2,3,4,4,5,5,6,6,7,
   8,8,9,9,1Ø,11,11, 12,12,13,13,14
5Ø3Ø DATA 15,15,16,16,17,18,18,19,19,
   2Ø,2Ø,21,22
5Ø4Ø DIM TB(37): FOR I=1 TO 37:
   READ TB(I): NEXT: RETURN
1ØØØØ DATA 13,4,17,4,18,4
1ØØØ2 DATA 2Ø,2Ø,13,4,17,4,18,4
1ØØØ4 DATA 2Ø,2Ø,13,4,17,4,18,4
1ØØØ6 DATA 2Ø,8,17,8,13,8,17,8
1ØØØ8 DATA 15,2Ø,17,4,17,4,15,4
1ØØ1Ø DATA 13,8,13,8,17,8,2Ø,8
1ØØ12 DATA 2Ø,4,18,2Ø,17,4,18,4
1ØØ14 DATA 2Ø,8,17,8,13,8,15,8
1ØØ16 DATA 13,2Ø
```

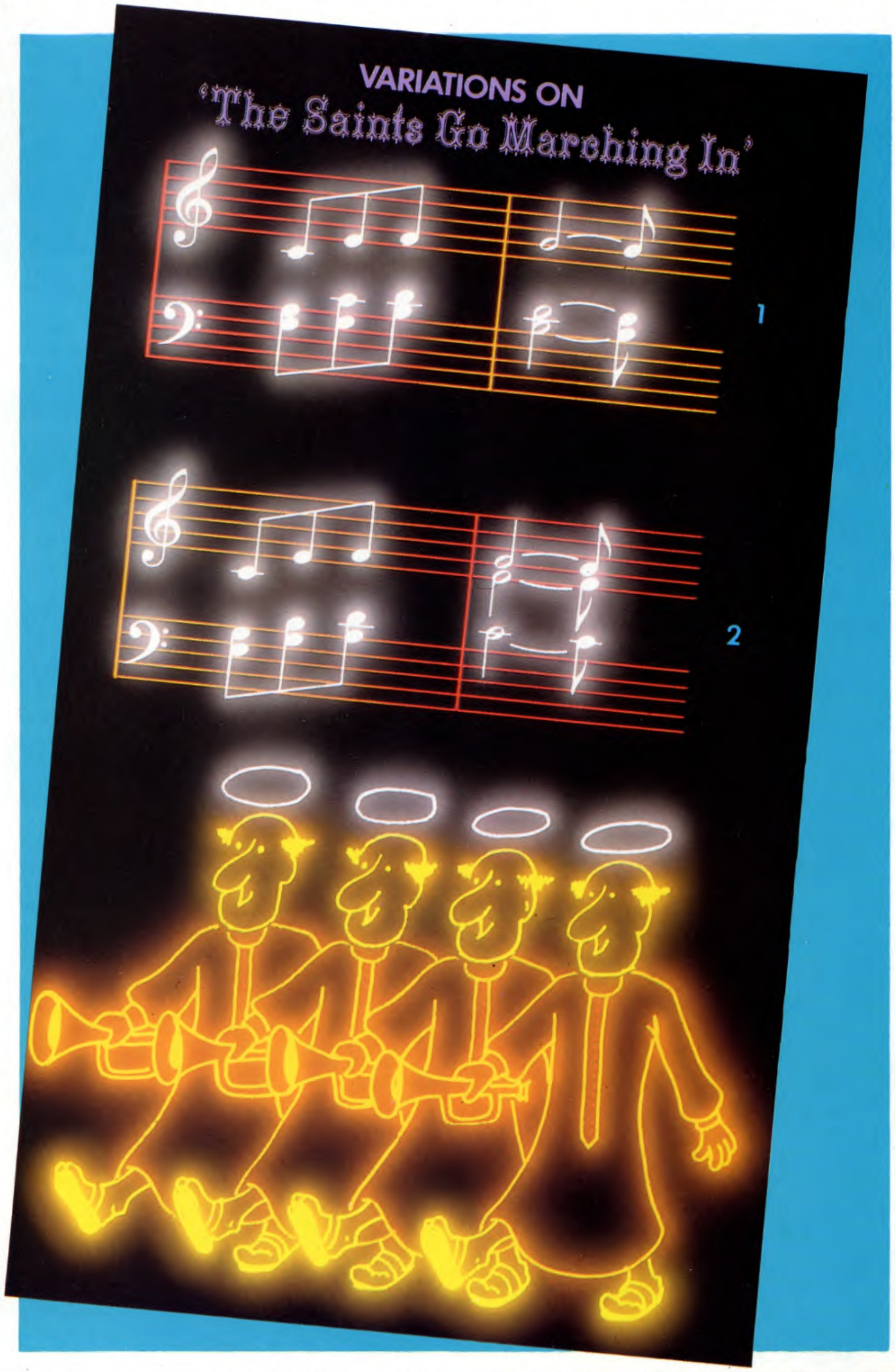
This is quite a complicated program, so fairly detailed notes are provided. The program shares, however, a certain amount of code with the programs given in the previous article (see pages 701–707).

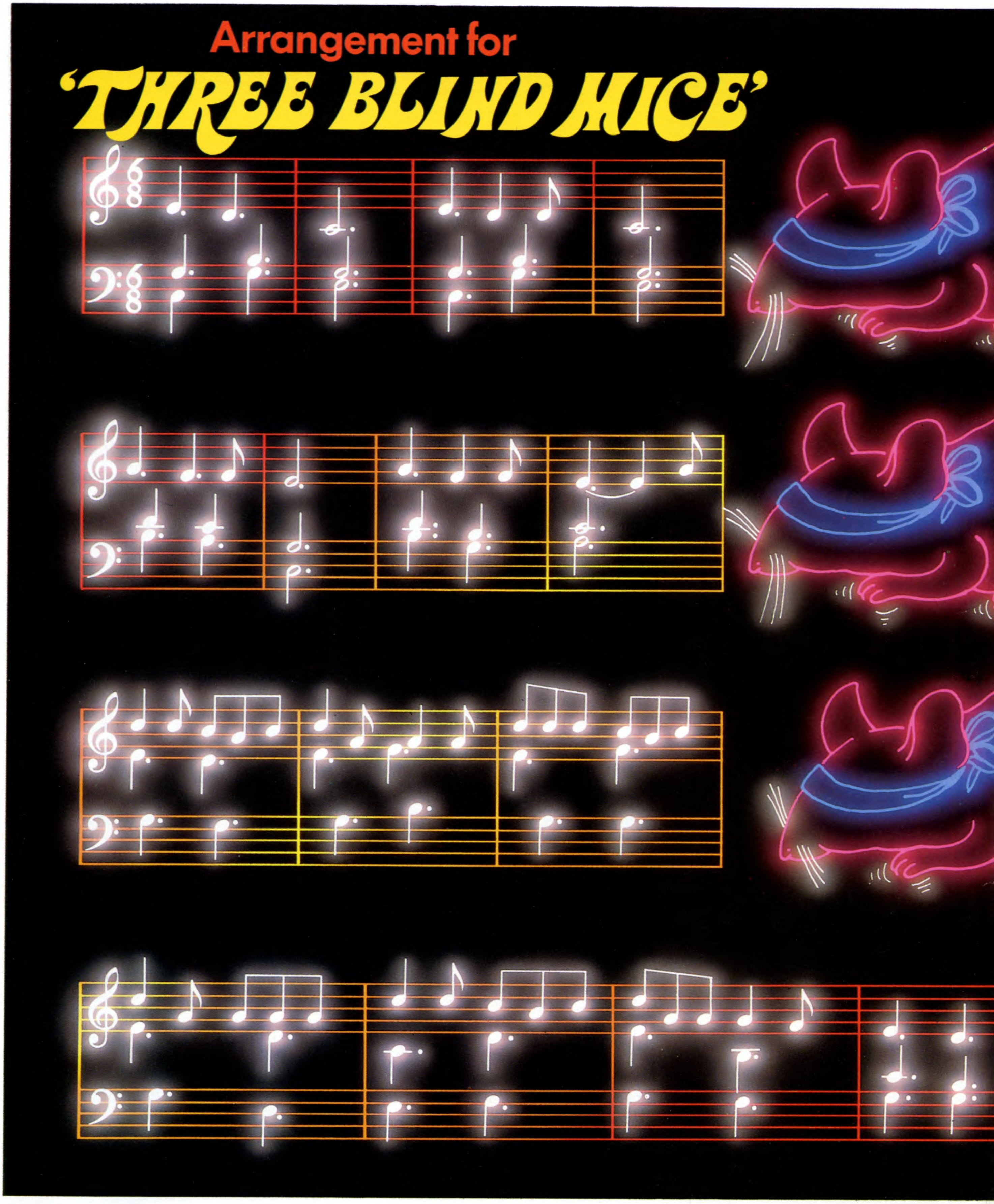
Lines 1Ø to 9Ø set up the program, inputting a 'tempo' value, calling three initialization subroutines, and setting variables KØ to K5 to the values Ø to 5. Variables are used rather than constants in the part of the program in which speed is critical (Lines 1ØØ–2ØØ), since variables can be handled faster than constants in the form of groups of digits. Spaces have been removed from this part of the program to increase speed although it does make it harder to read.

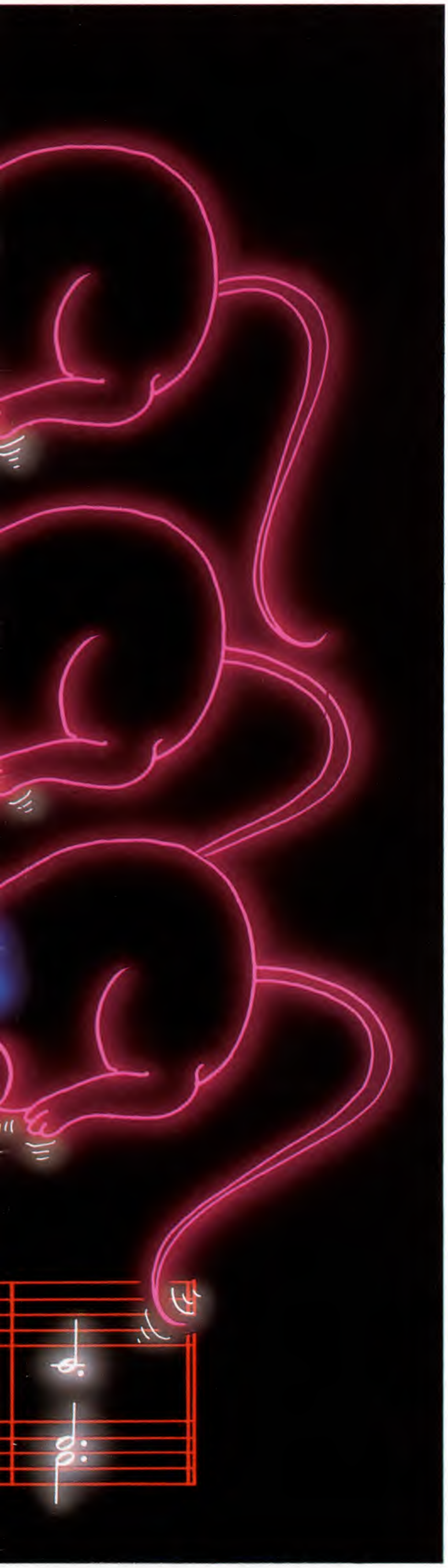
The subroutine at 3ØØØ initializes the SID chip, setting values for the attack, decay, sustain and release parameters for all three voices. The sustain level for voice one is higher than that for voices two and three: consequently voice one sounds louder than the other two voices, so the melody will sound louder than the accompaniment.

The subroutine at 4ØØØ sets up the high and low bytes in arrays LQ% and HQ% to control the frequencies of the 37 notes in the range of the instrument.

The subroutine at 5ØØØ sets up two arrays TA and TB that are used to define the scale of C major and to deduce appropriate chords to play with the melody. TA contains pitch numbers corresponding to the succession of 'white' notes in the key of C major, which could be called the note's 'scale number'. The




Arrangement for
'THREE BLIND MICE'



first note of the scale, with scale number 1, C, has pitch number 1, the second, D, has pitch number 3, the fourth, F, has pitch number 6 and so on. TB returns the scale number which corresponds to a given pitch number, so for example the pitch number 13 corresponds to the 8th note in the scale. The two arrays allow the pitch number to be extracted given the scale number, and vice versa: they allow opposite operations to be performed. These tables are needed by the logic that provides the automatic harmony.

The main loop is from 1ØØ–2ØØ. Line 1ØØ reads in the current pitch and duration values, and bypasses the next piece of processing if the note is a rest. Line 11Ø finds which scale number it is, using the array TB. Line 12Ø randomly generates either 2 or 3, which specifies how far below the melody the middle note in the harmonizing chord will be. If it's 3, then the other note, controlled by I2, must be 5 notes below the melody note. But if it's 2, then the bottom note may be either 4 or 5 notes below the melody note, and this is calculated by Line 13Ø.

This invariably produces one of the three possible triads that will harmonize with any note. For example, if the melody note is A, then the middle note may be F (2 notes in the scale below it) or E (3 notes below it), and the bottom note will be C or D. Consequently, the note A will harmonize with the chord of F major, D minor or A minor.

Line 135 sets I1 and I2 to the numbers of the scale numbers that are to be used. Lines 14Ø–16Ø POKE the high and low frequency registers with the appropriate values; the pitch for the melody is given by PV, which was read from the DATA statements. The pitch for the other two notes is derived using the array TA, which converts the number of the scale numbers into their pitch values. Lines 17Ø–19Ø switch the envelopes on, execute a delay loop, and switch the notes off at the end of the notes.

There is no test for the end of data as this would slow down the processing, so the program will give an 'out of data' error message when it has finished playing.

```
10 INPUT "TEMPO ", TP
20 GOSUB 5000
30 ENVELOPE 1,1,0,0,0,0,0,0,30,-2,
   0,0,120,100
40 ENVELOPE 2,1,0,0,0,0,0,0,30,-2,
   0,0,110,90
50 ENVELOPE 3,1,0,0,0,0,0,0,-127,-127,
   -127,-127,0,0
100 READ PV,T: IF PV=0 THEN 170
110 C=TB(PV)
120 I1=RND(2)+1: IF I1=3 THEN I2=5:
   GOTO 140
130 I2=RND(2)+3
140 SOUND &11,1,53+(PV-1)*4,-1
150 SOUND &12,2,53+(TA(C-I1)-1)
   *4,-1
160 SOUND &13,2,53+(TA(C-I2)-1)
   *4,-1
170 FOR DL=1 TO TP*T: NEXT
180 SOUND &11,3,0,-1
190 SOUND &12,3,0,-1
200 SOUND &13,3,0,-1
210 GOTO 100
5000 DATA 1,3,5,6,8,10,12,13,15,17,
   18,20,22,24,25,27,29,30,32,34,
   36,37
5010 DIM TA(22): FOR I=1 TO 22: READ
   TA(I): NEXT
5020 DATA 1,1,2,2,3,4,4,5,5,6,6,7,
   8,8,9,9,10,11,11,12,12,13,13,14
5030 DATA 15,15,16,16,17,18,18,
   19,19,20,20,21,22
5040 DIM TB(37): FOR I=1 TO 37:
   READ TB(I): NEXT: RETURN
10000 DATA 13,4,17,4,18,4
10002 DATA 20,20,13,4,17,4,18,4
10004 DATA 20,20,13,4,17,4,18,4
10006 DATA 20,8,17,8,13,8,17,8
10008 DATA 15,20,17,4,17,4,15,4
10010 DATA 13,8,13,8,17,8,20,8
10012 DATA 20,4,18,20,17,4,18,4
10014 DATA 20,8,17,8,13,8,15,8
10016 DATA 13,20
```

The logic of this has much in common with the Commodore 64 version, so the notes for that program should be read in conjunction with these. Line 2Ø calls the subroutine at 5ØØØ, which initializes the two arrays TA and TB, used to convert pitch numbers into scale numbers and vice versa.

Line 3Ø–5Ø define three envelopes: the first one is used by the melody, the second by the harmony (it is slightly quieter), and the third is used to switch the sound off between notes.

The main loop is from 1ØØ–21Ø. Line 1ØØ reads the next pitch and duration values, and bypasses some processing if the note is a rest. Line 11Ø deduces the scale number from the pitch number, 12Ø and 13Ø generate the scale numbers of the notes that will harmonize with the melody, and the notes are played by Lines 14Ø–16Ø. Line 17Ø is a delay, and the notes are switched off by Lines 18Ø–2ØØ.

Notice that the SOUND commands each have the second hex digit of their first parameter set to 1, and their last parameter set to −1. This ensures that each note continues until it is interrupted by a new note, and makes it easier to play notes simultaneously.

Q+A

Is there any way of adapting a home micro which is incapable of playing more than one note at a time to play chords or simultaneous melodies and harmonies.

Although the hardware on some home micros such as the Spectrum doesn't support music with more than one voice, it is possible to add hardware that will enable the Spectrum to do this. You can interface your computer with a synthesiser using MIDI, a recently announced standard for connecting home micros with certain keyboards.

The connection allows the computer's memory to be used to store notes and be used as a sequence—playing back sequences of notes and chords which can accompany live music.

The MIDI interfaces are now available for the Spectrum, Commodore 64 and the Acorn. Portable MIDI standard music keyboards will soon be available to accompany the microcomputer interfaces. These low-cost units are being manufactured specifically for home micro users.

SIMULTANEOUS MELODIES

With the previous program, all three musical parts change simultaneously—when a melody note changes, so does the accompanying chord—and also, only triads are played. The next program allows you to specify exactly what is played by all three parts. Any combination of notes can be played, and the melody can move at a different rate from the accompaniment.

As before, the music is held in DATA statements at the end of the program. The data for the whole of the first voice—from the beginning to the end—is stored first, then the data for the whole of the second, and then the whole of the third. For information on how to convert sheet music into the form suitable for this program read the previous music article (pages 701–707).

The DATA for each of the three voices is terminated by the special pair of values 99,99 (in Lines 1ØØ31, 2ØØ2Ø and 3ØØ2Ø in the data supplied). The other DATA statements each contain note definitions for exactly two bars, to make it relatively easy to determine which DATA statement contains note information for which bar. If you want to modify this program for a piece containing only two voices, immediately follow the 99,99 pair at the end of the second voice with another 99,99 pair, so the third voice is effectively assigned a null batch of data.

The programs include data for 'Three Blind Mice' in a three-part arrangement shown in the diagram on page 988. The harmony sticks mostly to simple triads: the first chord is C major, the second is G major and the third is C major again.

```
1Ø GOSUB 3ØØØ
2Ø INPUT "♥TEMPO";TP
3Ø GOSUB 4ØØØ
4Ø GOSUB 5ØØØ
5Ø KØ=Ø:K1=1:K2=2:K3=3
1ØØ P1=Ø: P2=Ø: P3=Ø
11Ø GOSUB 1ØØØ
12Ø GOSUB 11ØØ
13Ø GOSUB 12ØØ
14ØIFT1<>99THENT1=T1-K1:IFT1=
   KØTHENP1=P1+K2:GOSUB1ØØØ
15ØIFT2<>99THENT2=T2-K1:IFT2=
   KØTHENP2=P2+K2:GOSUB11ØØ
16ØIFT3<>99THENT3=T3-K1:IFT3=
   KØTHENP3=P3+K2:GOSUB12ØØ
17Ø FOR DL=1 TO TP: NEXT
18Ø GOTO 14Ø
1ØØØ POKE E1,EF:T1=DA%(K1, P1+K1)
1Ø1Ø PV=DA%(K1,P1):IF PV=99 OR
   PV=KØ THEN RETURN
1Ø2Ø POKE E1,EN:POKE H1,HQ%(PV):
   POKE L1,LQ%(PV)
1Ø3Ø RETURN
11ØØ POKE E2,EF:T2=DA%(K2,
   P2+K1)
111Ø PV=DA%(K2,P2):IF PV=99 OR
   PV=KØ THEN RETURN
112Ø POKE E2,EN:POKE H2,HQ%(PV):
   POKE L2,LQ%(PV)
113Ø RETURN
12ØØ POKE E3,EF:T3=DA%(K3,
   P3+K1)
121Ø PV=DA%(K3,P3):IF PV=99 OR
   PV=KØ THEN RETURN
122Ø POKE E3,EN:POKE H3,HQ%
   (PV):POKE L3,LQ%(PV)
123Ø RETURN
3ØØØ SI=54272
3Ø1Ø EN=33: EF=32
3Ø2Ø FOR I=SI TO SI+28: POKE I,Ø: NEXT
3Ø3Ø POKE SI+5,16*1+1
3Ø4Ø POKE SI+6,16*15+1
3Ø5Ø POKE SI+12,16*1+1
3Ø6Ø POKE SI+13,16*1Ø+1
3Ø7Ø POKE SI+19,16*1+1
3Ø8Ø POKE SI+2Ø,16*1Ø+1
31ØØ POKE SI+24,6
311Ø L1=SI: H1=SI+1:
   E1=SI+4
312Ø L2=SI+7:H2=L2+1:
   E2=SI+11
313Ø L3=SI+14:H3=L3+1:
   E3=SI+18
314Ø RETURN
4ØØØ DIM HQ%(37), LQ%(37)
4Ø1Ø TMP=2227:P2=2↑(1/12)
4Ø2Ø FOR I=1 TO 37
4Ø3Ø LQ%(I)=TMP-256*INT(TMP/
   256): HQ%(I)=TMP/256
4Ø4Ø TMP=TMP*P2
4Ø5Ø NEXT: RETURN
5ØØØ DIM DA%(3,1ØØØ)
5Ø1Ø FOR VN=1 TO 3: P=Ø
5Ø2Ø READ DA%(VN,P): READ
   DA%(VN,P+1)
5Ø3Ø P=P+2
5Ø4Ø IF DA%(VN,P-2)=99 THEN NEXT VN
5Ø5Ø IF VN<4 THEN 5Ø2Ø
5Ø6Ø RETURN
1ØØØØ DATA 17,6,15,6,13,12
1ØØØ2 DATA 17,6,15,6,13,12
1ØØØ4 DATA 2Ø,6,18,4,18,2,17,12
1ØØØ6 DATA 2Ø,6,18,4,18,2,17,1Ø,2Ø,2
1ØØØ8 DATA 25,4,25,2,24,2,22,2,24,2,
   25,4,2Ø,2,2Ø,4,2Ø,2
1ØØ1Ø DATA 25,2,25,2,25,2,24,2,22,2,
   24,2,25,4,2Ø,2,2Ø,2,2Ø,2,2Ø,2
1ØØ12 DATA 25,4,25,2,24,2,22,2,24,2,
   25,2,2Ø,2,2Ø,2,2Ø,4,18,2
1ØØ14 DATA 17,6,15,6,13,12
1ØØ2Ø DATA 99,99
2ØØØØ DATA 8,6,12,6,8,12
2ØØØ2 DATA 8,6,12,6,8,12
2ØØØ4 DATA 15,6,13,6,8,12
2ØØØ6 DATA 15,6,12,6,13,12
2ØØØ8 DATA 17,6,15,6,17,6,18,6
2ØØ1Ø DATA 17,6,18,6,17,6,15,6
2ØØ12 DATA 13,6,15,6,17,6,12,6
2ØØ14 DATA 13,6,12,6,5,12
2ØØ2Ø DATA 99,99
3ØØØØ DATA 1,6,8,6,5,12
3ØØØ2 DATA 1,6,8,6,5,12
3ØØØ4 DATA 12,6,1Ø,6,1,12
3ØØØ6 DATA 12,6,8,6,1Ø,12
3ØØØ8 DATA 8,6,6,6,8,6,12,6
3ØØ1Ø DATA 8,6,8,6,8,6,6,6
3ØØ12 DATA 5,6,6,6,8,6,6,6
3ØØ14 DATA 8,6,8,6,1,12
3ØØ2Ø DATA 99,99
```

Lines 1Ø to 5Ø set up the program, by inputting a 'tempo' value, calling several initialization subroutines, and setting variables KØ to K3 to the values Ø to 3. Variables are used rather than constants in the part of the program in which speed is critical, as with the previous program.

The subroutine at 3000 initializes the SID chip. Again the sustain level for voice one is higher than that for voices two and three for voice one to sound louder than the other two voices. Lines 3110–3130 set variables to the addresses of high and low frequency registers and envelope control registers for all three voices; again variables are used for speed.

The subroutine at 4000 sets up the high and low bytes in arrays LQ% and HQ% to control the frequencies of the 37 notes in the range of the instrument.

The subroutine at 5000 reads pitch and duration value pairs from the DATA statements into the array DA%(3.1000). The first index gives the voice number, the second is for the position in the array of the pitch and duration pair being handled. It is not possible to READ DATA directly from the DATA statements as the music is played, since the Basic interpreter supports only a single pointer that READs DATA statement items from the current position. As the logic of this program requires data to be read from different parts of the DATA statement region, the solution is to read the values from the DATA statements into the array mentioned, and this can then be read by 3 different pointers (P1, P2 and P3) which scan along the array as the music plays. If the different musical parts move with different rhythms, then the pointers will scan through the array at different rates. Line 5040 detects the value 99 that ends the block of data for each voice, advances the variable VN (for voice number) to its next value, and resets the pointer P to 0 (in Line 5010), to read in the block of data for the next voice.

The subroutine at 1000–1030 handles a new note in voice number 1; P1 points to the current pair of values specifying its pitch and duration. First the subroutine switches the envelope off, thus ending the previous note. It then fetches the duration data for the new note, which is put into the variable T1. Line 1010 assigns the pitch value of the current note to the variable PV; if it's 99, the end of the data for that voice has been reached, and if it's 0 then the note is a rest. In either case the subroutine is exited with a RETURN statement. Otherwise, Line 1120 switches on the envelope, and loads the high and low frequency registers, accessing the arrays LQ% and HQ% using the pitch value as an index. The subroutines at 1100 and 1200 perform the same operations for voices 2 and 3.

Line 100 sets the pointers that are to scan through the array DA%, and 110–130 call the subroutines at 1000, 1100 and 1200 to set up the first notes. Lines 140–180 constitute the main loop of the program. Line 140 tests T1, the duration value for the current note: a value of 99 indicates that the end of the voice data for that voice has been reached, and nothing more is done. If it is not 99, the value is decremented: if it becomes 0, it means that the end of the note has been reached and a new one is fetched by calling the subroutine at 1000. If the value hasn't yet reached 0, then nothing is done. Lines 150 and 160 perform similar operations for voices 2 and 3. Line 170 provides a delay loop which allows the tempo to be altered. Each time round the loop corresponds to a single time unit or clock pulse for the piece of music being played.

```
10 INPUT "TEMPO ", TP
20 GOSUB 5000
30 ENVELOPE 1,1,0,0,0,0,0,0,30,-2,0,
   0,120,100
40 ENVELOPE 2,1,0,0,0,0,0,0,30,-2,0,
   0,110,90
50 ENVELOPE 3,1,0,0,0,0,0,0,-127,-127,
   -127,-127,0,0
100 P1=0: P2=0: P3=0
110 GOSUB 1000
120 GOSUB 1100
130 GOSUB 1200
140 IF T1<>99 THEN T1=T1-1: IF T1=0
   THEN P1=P1+2: GOSUB 1000
150 IF T2<>99 THEN T2=T2-1: IF T2=0
   THEN P2=P2+2: GOSUB 1100
160 IF T3<>99 THEN T3=T3-1: IF T3=0
   THEN P3=P3+2: GOSUB 1200
170 FOR DL=1 TO TP: NEXT
180 GOTO 140
1000 SOUND &11,3,0,-1
1005 T1=DA(1,P1+1)
1010 PV=DA(1,P1): IF PV=99 OR PV=0
   THEN RETURN
1020 SOUND &11,1,53+(PV-1)*4,-1
1030 RETURN
1100 SOUND &12,3,0,-1
1105 T2=DA(2,P2+1)
1110 PV=DA(2,P2): IF PV=99 OR PV=0
   THEN RETURN
1120 SOUND &12,2,53+(PV-1)*4,-1
1130 RETURN
1200 SOUND &13,3,0,-1
1205 T3=DA(3,P3+1)
1210 PV=DA(3,P3): IF PV=99 OR PV=0
   THEN RETURN
1220 SOUND &13,2,53+(PV-1)*4,-1
1230 RETURN
5000 DIM DA(3,1000)
5010 FOR VN=1 TO 3: P=0
5020 READ DA(VN,P): READ DA(VN,P+1)
5030 P=P+2
5040 IF DA(VN,P-2)=99 THEN NEXT VN
5050 IF VN<4 THEN 5020
5060 RETURN
10000 DATA 17,6,15,6,13,12
10002 DATA 17,6,15,6,13,12
10004 DATA 20,6,18,4,18,2,17,12
10006 DATA 20,6,18,4,18,2,17,10,20,2
10008 DATA 25,4,25,2,24,2,22,2,24,2,
   25,4,20,2,20,4,20,2
10010 DATA 25,2,25,2,25,2,24,2,22,2,
   24,2,25,4,20,2,20,2,20,2
10012 DATA 25,4,25,2,24,2,22,2,24,2,
   25,2,20,2,20,2,20,4,18,2
10014 DATA 17,6,15,6,13,12
10020 DATA 99,99
20000 DATA 8,6,12,6,8,12
20002 DATA 8,6,12,6,8,12
20004 DATA 15,6,13,6,8,12
20006 DATA 15,6,12,6,13,12
20008 DATA 17,6,15,6,17,6,18,6
20010 DATA 17,6,18,6,17,6,15,6
20012 DATA 13,6,15,6,17,6,12,6
20014 DATA 13,6,12,6,5,12
20020 DATA 99,99
30000 DATA 1,6,8,6,5,12
30002 DATA 1,6,8,6,5,12
30004 DATA 12,6,10,6,1,12
30006 DATA 12,6,8,6,10,12
30008 DATA 8,6,6,6,8,6,12,6
30010 DATA 8,6,8,6,8,6,6,6
30012 DATA 5,6,6,6,8,6,6,6
30014 DATA 8,6,8,6,1,12
30020 DATA 99,99
```

The subroutine at 5000 loads the array DA: see the Commodore 64 program notes.

Lines 30–60 define the three envelopes that are used as in the previous program.

The subroutine at 1000 sets up the next note in voice one; first it forcibly switches the note off, using envelope 3, then it retrieves pitch and duration values using pointer P1 to the current data pair. If it's a rest, or if the end of the data for that voice has been reached then the subroutine is left. Otherwise the new note is started. The subroutines at 1100 and 1200 perform the same operations for the other two voices.

Lines 100–130 initialize the pointers that are to scan through the array, and set up the first 3 notes by calls to the subroutines at 1000, 1100 and 1200. The main loop is from 140–180. Line 140 tests the duration value for the current note of voice one: if it's 99 then the end of the voice data has been reached and nothing more is done. If it isn't 99, the value is decremented; if it becomes 0, then the end of the note has been reached and a new one is fetched by calling the subroutine at 1000. Lines 150 and 160 perform similar operations for voices 2 and 3. Line 170 provides a delay loop which allows the tempo to be altered. Each time round the loop corresponds to a single time unit or clock pulse for the piece of music being played.

CLIFFHANGER: BEGINNING THE GRAPHICS

Willie needs a cliff on which to perform his acts of daring-do. And you need to know how to couple together the bits of programming you have entered so far

It is now time to start drawing the cliff Willie has to scale. To put this on the screen you need a great deal of data. And again this is POKEd into a data table by a BASIC program. You will not be able to see the graphics at this stage—displaying them is covered by the next part. So for now, you just need to enter them and save the resulting data table.

By now you should have a number of the data POKEr programs and several machine code routines. And it is also time to start merging these together so that you can build them week by week into the whole game.

Now you need the data for the UDGs—the clouds, the seagulls, the boulders, the picnic goodies, the holes, the snakes and Willie himself. There is a lot of data here, but to get smooth animation, moving objects have to be drawn in several positions. The UDGs are then alternated to give the impression of continuous action.

```
5 CLEAR 56999
1Ø FOR n=57ØØØ to 57327: READ a:
   LET a$=STR$ a: POKE n,VAL
   ("BIN"+a$): NEXT n
9Ø1Ø DATA 11ØØØ,1111ØØ,1111ØØ,11ØØØ,
   1111ØØ,1111ØØ,1111ØØ,1111ØØ,1111ØØ,
   1111ØØ,11ØØØ,11ØØØ,11ØØØ,11ØØØ,
   11ØØØ,1111Ø
9Ø11 DATA 1,11,11,1,Ø,1,1,1,1ØØØØØØØ,
   11ØØØØØØ,11ØØØØØØ,1ØØØØØØØ,Ø,Ø,Ø,
   111ØØØØØ,111Ø,Ø,1,1Ø,1ØØ,1ØØØ,1ØØ,Ø,
   Ø,Ø,1ØØØØØØØ,1ØØØØØØ,1ØØØØØ,
   1ØØØØØ,11ØØØØ,Ø
9Ø12 DATA Ø,Ø,Ø,Ø,11ØØØ,1111ØØ,1111ØØ,
   11ØØØ,Ø,1ØØØØ,1ØØØØ,1111Ø,111ØØØØØ,
   Ø,11ØØ,1ØØ1ØØ,1ØØØØ1Ø,1ØØØØØ1Ø,
   1ØØØØ11,Ø,Ø,Ø,Ø,Ø
9Ø13 DATA Ø,Ø,Ø,Ø,1,11,11,1,Ø,Ø,Ø,Ø,
   1ØØØØØØØ,11ØØØØØØ,11ØØØØØØ,
   1ØØØØØØØ,Ø,1,1,1,111Ø,Ø,1,1Ø,Ø,Ø,Ø,
   111ØØØØØ,Ø,Ø,1ØØØØØØØ,1ØØØØØØ,1ØØ,
   1ØØØ,1ØØ,Ø,Ø,Ø,Ø,Ø,1ØØØØØ,1ØØØØØ,
   11ØØØØ,Ø,Ø,Ø,Ø,Ø
9Ø14 DATA 111ØØ,11111Ø,1111111,11111111,
   11111111,1111111Ø,111111ØØ,111ØØØ,11,
   111,1111,1111,1111,111,11,1,1ØØØØØØØ,
   11ØØØØØØ,111ØØØØØ,1111ØØØØ,
   1111ØØØØ,1111ØØØØ,111ØØØØØ,
   11ØØØØØØ
9Ø15 DATA Ø,Ø,111,11ØØØ,1ØØØØØ,
   1ØØØØØØ,1ØØØØØØ,1ØØØØØØØ,Ø,Ø,11111,
   1Ø1ØØØØØ,11ØØØØØØ,Ø,Ø,Ø,Ø,Ø,
   1ØØØØØØØ,1ØØØØØØ,1Ø111ØØ,1ØØØ1Ø,
   1Ø,1Ø,1ØØØØØØØ,1ØØØØØØ,11111ØØ,1Ø,
   1Ø,1,Ø,Ø,Ø,Ø,1ØØ,1Ø1Ø,1ØØØ1,
   111ØØØØØ,Ø,1Ø,1ØØ,1ØØØ,1ØØ,1ØØ,1ØØ,
   11111ØØØ,Ø
9Ø16 DATA Ø,Ø,1111ØØØ,1ØØØØ11Ø,1,1,Ø,Ø,
   Ø,Ø,1111Ø,11ØØØØ1,1ØØØØØØØ,
   1ØØØØØØØ,Ø,Ø,Ø,Ø,Ø,Ø,1ØØØØ111,
   1111ØØ1,Ø,Ø,Ø,Ø,Ø,Ø,111ØØØØ1,
   1ØØ1111Ø,Ø,Ø
9Ø17 DATA 1ØØØ1Ø,1Ø1ØØ,1ØØØ,1ØØØ,1ØØØ,
   1ØØØ,1ØØØ,1ØØØ,11ØØØ,1111ØØ,11Ø11Ø,
   11111Ø,11111Ø,1111ØØ,11ØØØ,11ØØØ,
   11ØØØ,11ØØØ,11ØØ,11ØØ,11Ø,11Ø,11,11,
   11Ø,11Ø,11ØØ,11ØØ,11ØØØ,11ØØØ,
   11ØØØØ,11ØØØØ,11ØØØØØ,11ØØØØØ,
   11ØØØ11Ø,11ØØØØ11,11ØØ11Ø,11Ø11ØØ,
   111ØØØ,111ØØØ
9Ø18 DATA 1ØØØØ1ØØ,11Ø1Ø11Ø,11111111,
   11111111,11111111,11111111,11111111,
   11111111,Ø,1,11,111,11111,111111,
   1111111,11111111
9Ø19 DATA Ø,Ø,11111111,11111111,1111ØØ,
   1111ØØ,11111111,11111111,11Ø,1ØØØ,
   111Ø11Ø,11111111,11111111,11111111,
   111111Ø,1111ØØ,1ØØØØ,1ØØØØ,1ØØØØ,
   111ØØØ,111ØØØ,111ØØØ,111ØØØ,111ØØØ,
   1ØØØØ,111ØØØ,11111ØØ,111ØØØ,111ØØØ,
   111ØØØ,1ØØØØ,1ØØØØ
9Ø2Ø DATA Ø,Ø,Ø,1ØØØØØ,1Ø1ØØØ1,
   1ØØØ1Ø1Ø,1ØØ,Ø,Ø,Ø,Ø,1ØØØØØ1Ø,
   1ØØØ1Ø1,1Ø1ØØØ,1ØØØØ,Ø
```

RUN the program and SAVE the resulting machine code data table using the instruction:

```
SAVE "Cliff4" CODE 57ØØØ,327
```

GETTING IT ALL TOGETHER

Long programs often have to be written and assembled in bits. This then gives you the problem of merging the bits together to give one long working program. And just because the various parts work separately, this does not mean the whole thing will work when it is put together.

One of the main problems is overwriting. If you have SAVEd a few too many bytes there is a danger that you will overwrite part of the next routine. If you are SAVEing your assembled routines using the machine code monitor you have to supply the start address and the number of bytes you want to SAVE. Or if you use the Spectrum's own machine code SAVE command you use the format:

```
SAVE "name" CODE
```

which again is followed by the start address, a comma, then the number of bytes to be SAVEd.

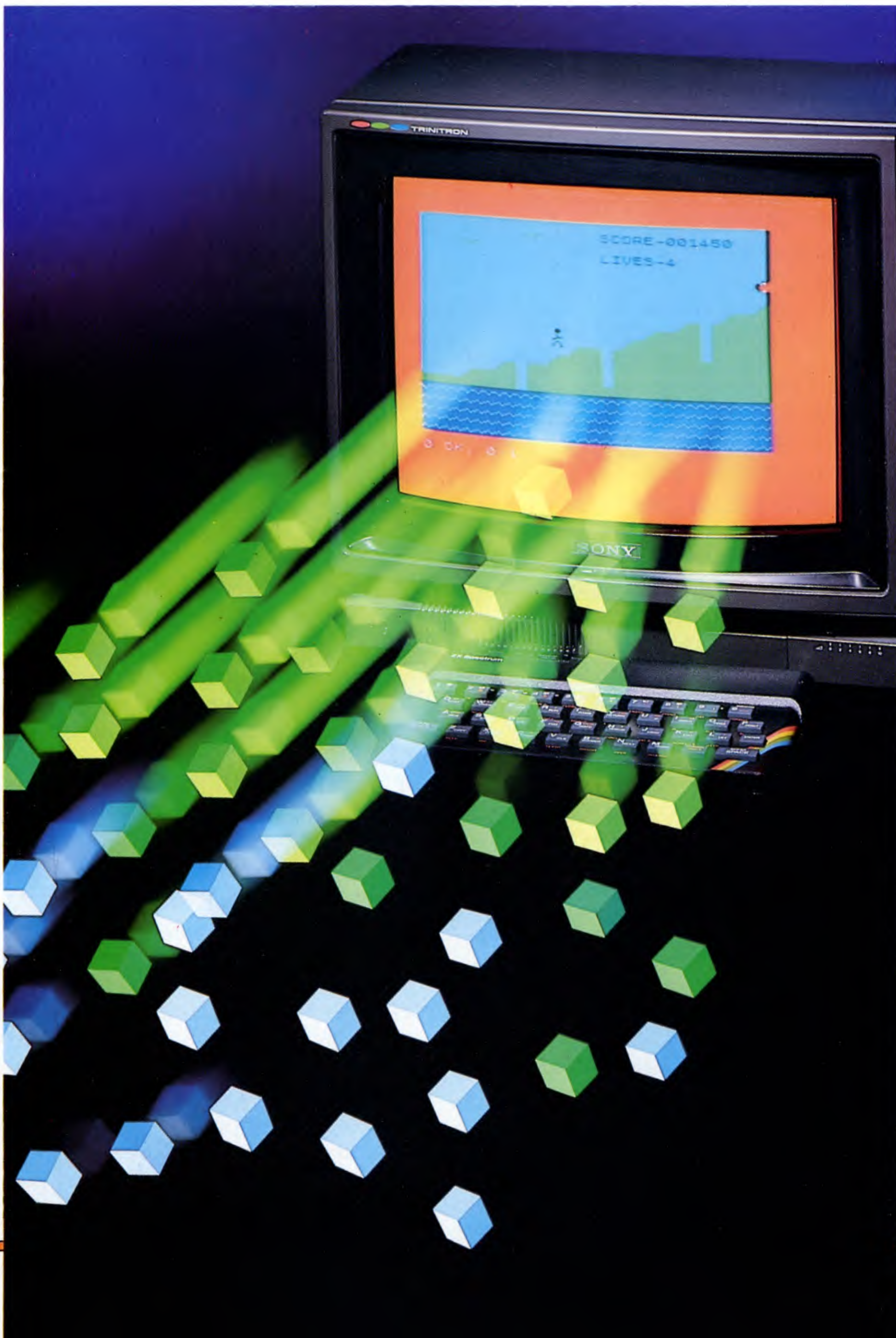
Obviously there is no problem with the start addresses. With machine code assembled from assembly language programming the start address of the object code is the same as the origin. After all, when the assembler is RUN it takes the origin and starts assembling the machine code program from there. And with data tables POKEd in from BASIC, the start address is the initial value of the variable in the FOR ... NEXT loop which picks up the DATA and stores it in memory a byte at a time when the program is RUN.

But working out the number of bytes to be SAVEd can cause problems. The end address given by assemblers is often the first free address past the end of the program. And

- THE PROFILE OF THE SLOPE
- ADDING UDG AND SPRITE **DATA**
- SIMULATING SMOOTH ANIMATION
- **LOAD**ING PROBLEMS
- **SAVE**ING EXTRA BYTES

sometimes all you are given is the start address of the last instruction, so if the routine does not end with a **ret**, it does not give a true end address.

The usual way round this problem is to SAVE an extra couple of bytes. But that can cause overwriting problems.



The answer is to LOAD your programs back into the computer in order up memory. LOAD the one with the lowest origin or start address first, then the next lowest, then the next and so on. This means that any extra bytes SAVEd at the end of the routine will be overwritten by the beginning of the next program—instead of the other way round. And it means that any **ret** that has been added to allow you to test single routines will be overwritten too.

When all the routines and data tables have been LOADed, SAVE them back to tape or Microdrive as one program under another name—the start address will be the start address of the first program, and the end address will be the end address of the last program. Then, if you have any problem you can put the various pieces back together and try again. Using the SAVE option on your assembler don't forget to SAVE the assembly language and the BASIC POKEr programs as well in case you need to re-assemble them later.

If you have any problems with the data tables, you can LOAD up the machine code routines—in order up memory again—and LOAD up and RUN the BASIC POKEr programs again. These must be LOADed one at a time, RUN and NEWed before you LOAD the next one. Again the whole area of memory should be SAVEd to tape under a new program name.

The following program supplies the data for the sprites—the clouds, the seagulls, the boulders, the picnic goodies, the holes, the snakes and Willie himself. It may look like there is a lot of data here, but to get smooth animation, moving objects have to be drawn in several positions. The sprites are then alternated to give the impression of continuous action.

```
20 V=53248
30 S=230
40 FORI=S*64 TO(S+8)*64
50 READA:POKEI,A:NEXT
60 POKEV+21,1:POKEV,100:POKEV+1,
   100:POKEV+39,0:POKE2040,237
100 PRINT"♥FINISHED":STOP
1000DATA0,0,0,0,0,0,0,0,0,0,0,0,0,7,
   128,0,7,64,0,7,128,0,7,128,0,15,192,
   0,11
1010DATA64,0,11,64,0,11,64,0,12,192,0,7,
   128,0,3,0,0,3,0,0,3,0,0,3,0,0,3
1020DATA0,0,3,128,0,3,192,0
1040DATA0,0,0,0,0,0,0,0,0,0,0,0,0,7,
```

```
   128,0,7,64,0,7,128,0,7,128,0,15,192,
   0,11
1050DATA64,0,6,96,0,28,112,0,56,216,
   0,7,128,0,3,128,0,3,192,0,31,96,0,
   31,96
1060DATA0,24,56,0,16,60,0,0,0,0,0
1080DATA0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0
1090DATA30,0,0,63,0,0,94,128,0,237,
   192,0,243,192,0,243,192,0,237,192,0,
   94,128
1100DATA0,63,0,0,30,0,0,0
1120DATA0,0,0,0,0,0,15,192,0,245,96,
   15,170,160,245,85,96,255,170,224,
   255
1130DATA245,224,255,255,224,143,255,
   224,128,255,224,240,15,160,255,0,
   32,255
1140DATA240,96,255,255,224,15,255,
   224,0,255,192,0,15,128,0,0,0,0,0,0,
   0,0,0,0
1160DATA0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,24,0,0,60,0,0,102,0,0,60,0,0,
   120,0
1170DATA120,0,0,120,0,0,60,0,0,30,
   0,0,30,0,0,30,0,0,60,0,0,120,0,0,
   120,0,0
1180DATA48,0,0,32,0,0,0
1200DATA36,0,0,24,0,0,8,0,0,16,0,0,8,
   0,0,24,0,0,60,0,0,102,0,0,60,0,0,
   120,0
1210DATA0,120,0,0,120,0,0,60,0,0,30,
   0,0,30,0,0,30,0,0,60,0,0,120,0,0,
   120,0,0
1220DATA48,0,0,32,0,0,0
1240DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,255,0,0,255,0,0,255,0,0,
   255
1250DATA0,0,255,0,0,255,0,0,255,0,0,
   255,0,0,255,0,0,255,0,0,255,0,0,
   255,0,0
1260DATA255,0,0,255,0,0,255,0,0,0
1280DATA0,0,0,0,0,0,0,0,0,192,0,1,32,
   192,2,27,48,26,4,8,36,8,8,66,8,16,
   130
1290DATA4,96,132,0,144,128,0,8,64,0,8,
   68,43,48,56,65,64,8,64,128,7,35,0,
   0,220
1300DATA0,0,0,0,0,0,0,0,0,0,0,0
```

The following program provides the user-defined graphics which make up the characters that stand still. It also defines the letters of the alphabet. This gives you a unique typeface for your game. Although this is not entirely necessary, it is often done in commercial games.

```
20FORI=0TO263:READA:POKE12288
   +I,A:NEXT
30FORI=0TO127:READA:POKE12672
   +I,A:NEXT
40PRINT"♥FINISHED":STOP
1000DATA60,66,153,145,145,153,
   66,60
1010DATA56,68,130,130,254,130,
   130,0
1020DATA248,132,130,252,130,130,
   252,0
1030DATA124,130,128,128,128,130,
   124,0
1040DATA248,132,130,130,130,132,
   248,0
1050DATA248,132,128,248,128,132,
   248,0
1060DATA248,132,128,248,128,128,
   128,0
1070DATA120,132,128,152,132,132,
   120,0
1080DATA132,132,132,252,132,132,
   132,0
1090DATA56,16,16,16,16,16,56,0
1100DATA56,16,16,16,16,144,112,0
1110DATA132,136,144,224,144,136,
   132,0
1120DATA128,128,128,128,128,128,
   252,0
1130DATA130,198,170,146,130,130,
   130,0
1140DATA130,194,162,146,138,134,130,0
1150DATA124,130,130;130,130,130,124,0
1160DATA124,130,130,252,128,128,128,0
1170DATA124,130,130,252,136,132,130,0
1180DATA124,130,130,252,136,132,130,0
1190DATA120,132,128,52,12,132,120,0
1200DATA254,146,16,16,16,16,16,0
1210DATA130,130,130,130,130,130,
   124,0
1220DATA130,130,130,130,68,40,16,0
1230DATA130,130,146,146,146,146,
   124,0
1240DATA130,68,40,16,40,68,130,0
1250DATA130,68,40,16,16,16,16,0
1260DATA254,4,8,16,32,64,254,0
1270DATA56,56,16,254,40,68,130,0
1280DATA170,170,170,255,255,255,255,
   255
1290DATA3,7,15,31,31,63,63,127
1300DATA223,239,255,255,239,247,255,
   255
1310DATA223,239,255,255,239,247,255,
   255
1320DATA0,0,0,0,0,0,0,0
1470DATA124,130,130,130,130,130,
   124,0
1480DATA16,48,16,16,16,16,56,0
1490DATA124,130,2,124,128,128,254,0
1500DATA124,130,2,124,2,130,124,0
1510DATA144,144,144,252,16,16,16,0
1520DATA252,128,128,248,4,4,252,0
1530DATA60,64,128,248,132,132,252,0
1540DATA252,4,4,8,16,32,64,0
1550DATA120,132,132,120,132,132,120,0
1560DATA120,132,132,124,4,4,120,0
1590DATA0,0,36,90,153,0,0,0
1600DATA0,0,231,24,24,0,0,0
1610DATA129,66,36,24,24,0,0,0
1620DATA0,0,231,24,24,0,0,0
1630DATA34,85,132,0,34,85,132,0
1640DATA223,187,183,219,191,219,219,239
```

RUN these programs, then SAVE the machine code data table it makes, using your machine code monitor.

PICKING UP THE PIECES

Long programs often have to be written and assembled in bits. This then gives you the problem of merging the bits together to give one long working program. And just because the separate parts work, this does not mean the whole thing will work when it is put together.

When one routine calls another, there is always the problem that it calls it in the wrong place. And data tables may not coincide properly with the appropriate data pointers. But these problems can be eliminated by checking the individual sections programs thoroughly.

This assumes that when you have finished merging the parts of the program together you haven't lost anything in the process. Unfortunately, it is all to easy to lose a byte or two when you are concatinating various different programs in memory. And the loss of even a single byte can cause the whole program to crash or malfunction.

One of the main problems is overwriting. If you have a few too many bytes there is a danger that you will overwrite part of the next routine. If you are SAVEing your assembled routines using the machine code monitor you have to supply the start address and the number of bytes you want to SAVE.

Obviously, there is no problem with the start addresses. With machine code assembled from assembly language programming the start address of the object code is the same as the origin. After all, when the assembler is RUN it takes the origin and starts assembling the machine code program from there. And with data tables POKEd in from BASIC, the start address is the initial value of the variable in the FOR ... NEXT loop which picks up the DATA and stores it in memory a byte at a time when the program is RUN.

But working out the number of bytes to be SAVEd can cause problems. The end address given by assemblers is often the first free address past the end of the program. And sometimes all you are given is the start address of the last instruction, so if the

routine does not end with a RTS, it does not give a true end address.

The usual way round this problem is to SAVE an extra couple of bytes. But that can cause overwriting problems.

The answer is to LOAD your programs—using the LOAD"name",1,1 command—back into the computer in order up memory. LOAD the one with the lowest origin or start address first, then the next lowest, then the next and so on. This means that any extra bytes SAVEd at the end of the routine will be overwritten by the beginning of the next program—instead of the other way round. And it means that any RTS that has been added to allow you to test single routines will be overwritten too.

When all the routines and data tables have been LOADed, SAVE them back to tape or disk as one program under another name—the start address will be the start address of the first program and the end address will be the end adrdress of the last program. Then if you have any problem you can put the various pieces back together and try again. Using the SAVE option on your assembler don't forget to SAVE the assembly language and the BASIC POKEr programs as well in case you need to re-assemble them at a later stage.

If you have any problems with the data tables, you can LOAD up the machine code routines—in order up memory again—and LOAD up and RUN the BASIC POKEr programs again. These must be LOAded one at a time, RUN and NEWed before you LOAD the next one. Again the whole area of memory should be SAVEd to tape under a new program name.

The following program POKEs the data for the UDGs—the clouds, the seagulls, the boulders, the picnic goodies, the holes, the snakes and Willie himself—into memory. There is a lot of data here, but to get smooth animation, moving objects have to be drawn in several positions. The UDGs are then alternated to give the impression of continuous action. Each character is made up of eight bytes of data, and there are 16 bytes of DATA in each DATA line. The last figure in each line is a checksum which is totalled in Line 900. The program will check that the DATA adds up and will tell you which line an error has occurred in if you have keyed the DATA in wrongly.

If you get an error message without a line number, you have missed out a line completely. As always, type PAGE=&3000 and NEW before you key the program in.

```
60 DATA"FFFFFFFFFFFFFFFF",2040
70 DATA"183C3C1800000000",168
80 DATA"000000003C3C3C3C",240
90 DATA"3C3C000000000000",120
100 DATA"0000000000080808",24
110 DATA"1000181818181838",192
120 DATA"0000000000080878",136
130 DATA"030018244241C200",388
140 DATA"3C3C181818181838",296
150 DATA"030301020404040C",33
160 DATA"4231C20000000000",309
170 DATA"FF3E181818181838",493
180 DATA"FF63C20000000000",548
190 DATA"183C6E7E7E3C1818",554
200 DATA"2214080808080808",102
210 DATA"00DDFFFFFFFFFFFF",1751
220 DATA"0001070F1F3F3F7F",307
230 DATA"0080E0F0F8FCFCFE",1598
240 DATA"1C3E7FFFFFFE7C38",1161
250 DATA"387CFEFFFF7F3E1C",1161
260 DATA"00EE110000EE1100",510
270 DATA"0078860101000000",256
280 DATA"001E618080000000",383
290 DATA"0000008779000000",256
300 DATA"000000E19E000000",383
310 DATA"0000000000101010",48
320 DATA"080018181818181C",156
330 DATA"000000000010101E",62
340 DATA"C000182442824300",515
350 DATA"3C3C18181818181C",268
360 DATA"0F03010204080400",37
370 DATA"4282430000000000",263
380 DATA"FF3E18181818181C",465
390 DATA"FFA2430000000000",484
400 DATA"0000071820404080",319
410 DATA"00001FA040000000",255
420 DATA"000080405C220202",322
430 DATA"80403E0202010000",259
440 DATA"000000040A11E000",255
450 DATA"020408040404F800",274
460 DATA"0000000081818183",518
470 DATA"C3C3C7C7E7EFEFEF",1736
480 DATA"0810080402040201",45
490 DATA"0204080408102010",90
500 DATA"2040804020402040",480
510 DATA"2010201000100010",128
520 DATA"1008040804020102",45
530 DATA"0402040810081020",90
540 DATA"4020408040204020",480
550 DATA"1020102010001000",128
560 DATA"0000FFFFFF000000",765
570 DATA"000000FFFFFF0000",765
580 DATA"00000000007F7F00",254
590 DATA"0000000000FEFE00",508
600 DATA"007F7F0000000000",254
610 DATA"00FEFE0000000000",508
620 DATA"000000000000007F",127
630 DATA"00000000000000FE",254
640 DATA"7F00000000000000",127
650 DATA"FE00000000000000",254
660 DATA"000000003C7EFFFF",696
670 DATA"FFFFFFFF7F7F3F0F",1352
680 DATA"FFFFFFFFFEFEFCF0",2020
690 DATA"0000000101010000",3
700 DATA"3878C08080800000",752
710 DATA"1818181824428181",456
720 DATA"81818181818181FF",1158
730 DATA"007E7E7E7E7E7E00",756
740 DATA"0000010103000000",5
750 DATA"20F8FCFCFE000000",1038
760 DATA"0000000000030101",5
770 DATA"0000000000FEFCFC",758
780 DATA"F8F8F87070702020",1144
790 DATA"0000017F7F7F3C80",570
800 DATA"0007FCF0C0000000",691
810 DATA"80C0F07030000000",720
820 DATA"000000008080C070",560
830 DATA"0000000000030F3F",81
840 DATA"70330F8FCCF07030",925
850 DATA"FCF0C00000000000",684
860 DATA"000000000000030F",18
870 DATA"0000030F3FFCF0C0",765
880 DATA"0F0C0000030F0F0C",72
890 DATA"030F3FFCF0C00000",765
900 DATA 40941
940 P%=&1534
950 S%=0
960 FORA%=60TO890STEP10
970 READA$,B%
975 IFLEN A$<>16 THEN B%=0: GOTO
   1030
980 T%=0
990 FORC%=0TO7
1000 ?(C%+P%)=EVAL("&"+
   MID$(A$,C%*2+1,2))
1010 T%=T%+?(C%+P%)
1020 NEXT
1030 IFT%<>B% PRINT"Data error in
   line□";A%:END
1040 S%=S%+T%
1050 P%=P%+8
1060 NEXT
1070 READB%
1080 IFB%<>S% PRINT"Data error"
```

RUN this program to POKE the DATA into memory. *SAVE the resulting machine code data table using the instruction:

```
*SAVE "MCliff4"1534□17E0
```

PUTTING PROGRAMS TOGETHER

Long programs often have to be written and assembled in bits. This then gives you the problem of merging the bits together to give one long working program. And just because the separate parts work, this does not mean the whole thing will work when it is put together.

One of the main problems is overwriting. If you have a few too many bytes there is a danger that you will overwrite part of the next routine. If you are ★SAVEing the object code you have to supply the start address and the end address—or the number of byte with a

plus sign in front—of the machine code you want to ★SAVE.

Obviously there is no problem with the start addresses. With machine code assembled from assembly language programming the start address is the same as the origin. The origin is given in the P% value just before the square bracket switches on the assembler. When the assembler is RUN this value is used at start address for the machine code. And with data tables POKEd in from BASIC, the start address is the initial value of the variable in the FOR ... NEXT loop which picks up the DATA and stores it in memory a byte at a time when the program is RUN.

But working out the number of bytes to be ★SAVEd can cause problems. No end address is given by the Acorn's assembler. You can work out the end address of course. When the assembler is RUN the memory addresses that each instruction is put in appears on the screen. So you will be able to read the start address of the last instruction. You can then count the number of bytes the machine code equivalent of the last instruction takes and add it on to give the end address.

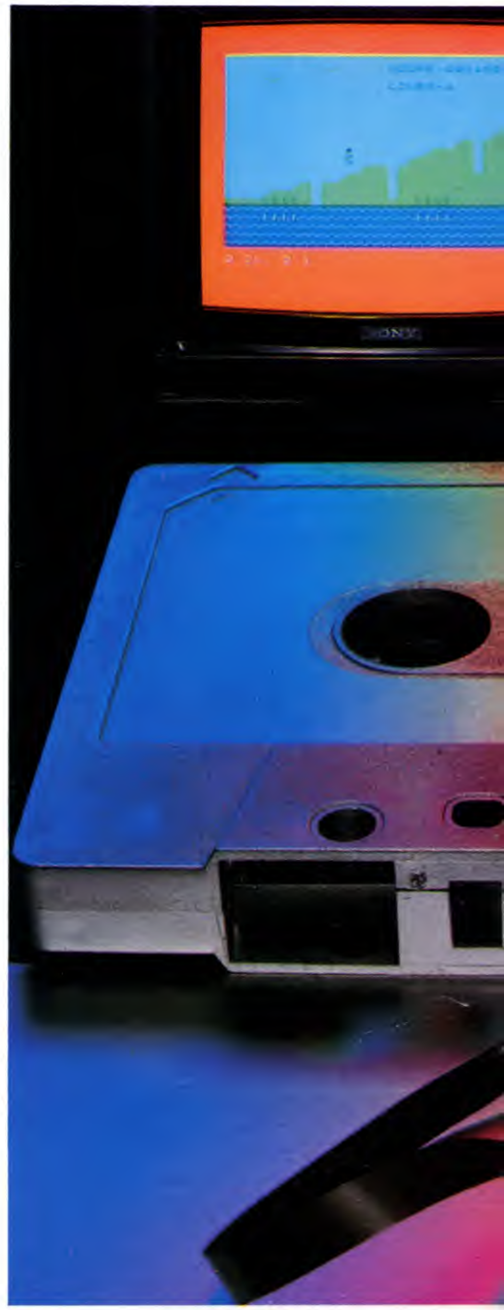
But it is usual to ★SAVE a couple of extra bytes to be on the safe side. Unfortunately this can cause overwriting problems. These extra bytes may overwrite the beginning of the program that follows.

The answer is to ★LOAD your programs back into the computer in order up memory. ★LOAD the one with the lowest origin or start address first, then the next lowest, then the next and so on. This means that any extra bytes ★SAVEd at the end of the routine will be overwritten by the beginning of the next program—instead of the other way round. And it means that any RTS that has been added to allow you to test single routines will be overwritten too. But don't forget to type in PAGE=&3000 and NEW before you ★LOAD your machine code programs back.

When all the routines and data tables have been ★LOADed, ★SAVE them back to tape or disk as one program under another name—the start address will be the start address of the first program and the end address will be the end address of the last program. Then if you have any problem you can put the various pieces back together and try again. Don't forget to SAVE the assembly language and the BASIC POKEr programs as well in case you need to re-assemble them later.

Add the following program lines to the BASIC data POKEr you've been building up.

```
110 READ A$
120 FOR A=1 TO LEN (A$)
130 POKE AD,ASC(MID$(A$,A,1))
140 AD=AD+1
150 NEXTA
160 DATA ###!##!###!##!
    ####!##!##!###!##!!
170 FORA=1 TO 702
180 READA$:POKEAD,VAL("&H"+A$)
190 AD=AD+1:NEXTA
200 IF AD<>18238 THENPRINT"ERROR"
210 DATA 55,54,50,50,40,40,0,0,7F,5F,57,
    D7,F5,FF,D5,75,DD,77,5D,D5,7D,75,5D,77,
    F5,FD,57,75,DD,77,5D,D5,FD,5F,57,D7,5D,
    FF,D5,7F,75,77,FD,F7,D5,5D,75,77,55,D5,
    D5,5D,5D,D7,F5,7D,D5,5D,5D,D7,55,57,
    FF,7F,57,57,FD,FD
220 REM sun
230 DATA 55,75,D5,55,55,75,D5,55,75,75,D5,
    D5,5D,5D,D5,D5,5D,5D,D7,55,57,5D,D7,
    5D,D5,D5,5D,5D,75,D7,DD,75,7D,5D,75,
    D5,5D,75,5D,D7,57,75,5D,5D,F5,D5,57,75,
    5D,D5,57,75,57,55,55,D7,F7,55,55,DD
240 DATA 57,55,55,D5,77,55,55,DF,D5,D5,57,
    55,5D,D5,57,7F,F5,75,5D,55,57,75,5D,F5,
    57,5D,75,5F,5D,57,D7,55,75,75,57,55,55,
    77,75,D5,55,D7,75,D5,55,D7,75,75,57,57,
    5D,75,57,57,5D,55,55,57,5D,55
250 REM numbers
260 DATA 7D,D7,D7,D7,7D,5D,7D,5D,5D,FF,
    7D,D7,5D,75,FF,FD,57,FD,57,FD,5D,7D,
    DD,FF,5D,FF,D5,7D,57,FD,7F,D5,FD,D7,
    7D,FF,57,5D,75,75,7D,D7,7D,D7,7D,7F,D7,
    7F,57,57
270 REM graphics
280 DATA 57,5F,5F,57,5F,5F,5F,5F,D5,F5,F5,
    D5,F5,F5,F5,F5,5F,5F,57,57,57,57,57,57,
    F5,F5,D5,D5,D5,D5,D5,FD,55,55,55,55,55,
    55,55,55,57,5F,5F,57,55,57,57,57,D5,F5,
    F5,D5,55,55,55,FD,55,55,55,55,55,55,55,
    55,55,55,55,55,55,55,55,55,FD,55,57,5D,
    75,D5,75,55
290 DATA 55,55,D5,75,5D,5D,5F,55,55,55,55,
    55,55,55,55,55,55,55,55,55,57,5F,5F,57,55,
    55,55,55,D5,F5,F5,D5,55,57,57,57,FD,55,
    57,5D,55,55,55,FD,55,55,D5,75,75,D5,75,
    55,55,55,55,55,5D,5D,5F,55,55,55,55,55,
    55,55,55,55,55,55,55,55,55,55,55,55,57,5F,
    5F,57
300 DATA 55,55,55,55,D5,F5,F5,D5,55,55,55,
    55,55,55,55,55,55,55,55,55,55,55,55,55,55,
    57,57,57,FD,55,57,5D,55,55,55,FD,55,55,
    D5,75,55,55,55,55,55,55,55,55,55,55,55,
    55,55,55,55,55,75,D5,75,55,55,55,55,55,
    5D,5D,5F,55,55,55,55,55,55,55
310 DATA 55,55,55,55,55,55,57,5F,7F,FF,FF,
    FF,7F,5F,F5,FD,FF,FF,FF,FD,F5,D5,55,55,
    55,55,55,55,55,55,5F,7F,FF,FF,FF,7F,5F,57,
    D5,F5,FD,FF,FF,FF,FD,F5,55,55,55,55,55,
    55,55,55,5D,57,55,55,55,55,55,55,5D,75,
    D5,D5,D5,D5,D5,D5,57,5F,7D,7F,7F,5F,57,
    57,D5,F5
```



```
320 DATA FD,FD,FD,F5,D5,D5,57,57,55,55,55,
    55,55,55,D5,D5,F5,F5,7D,7D,5F,5F,55,55,
    55,55,57,57,5F,5F,7D,7D,F5,F5,D5,D5,55,
    55,7D,7D,F5,F5,7D,7D,5F,5F,55,55,7D,5F,
    7D,F5,D5,D5,55,55,AA,AA,56,56,AA,AA,
    55,55,AA,AA,95,95,AA,AA,55,55,7F,FF,FF,
    FF,7F,5F,81
330 DATA 15,7D,FF,FF,FF,FD,F5,57,57,57,5F,
    5F,5F,5F,5F,55,55,55,D5,D5,D5,D5,D5,57,
    5F,7F,5F,5F,5F,57,57,55,D5,F5,D5,D5,D5,
    55,55,AA,AA,AA,A6,99,6A,AA,AA,AA,AA,
    AA,AA,A9,66,9A,AA,AA,AA,AA,6A,9A,A6,
    A9,AA,AA,AA,AA,A6,99,6A,AA,AA,
```

The hashes and exclamation marks in Line 160 define the silhouette of the cliff. A #



means a flat section and a ! means a slope.

Lines 21∅ to 33∅ supply the data needed for the UDGs—the sun, the boulders, the picnic goodies, the holes, the snakes and Willie himself. It may look like there is a lot of data here, but to get smooth animation moving objects have to be drawn in several positions. The UDGs are then alternated to give the impression of continuous action.

THE SUM OF THE PARTS

Long programs often have to be written and assembled in bits. This then gives you the problem of merging the bits together to give one long working program. And just because the separate parts work, this does not mean the whole thing will work when it is put together.

One of the main problems is overwriting. If you have a few too many bytes there is a danger that you will overwrite part of the next routine. If you are CSAVEing your assembled routines using the machine code monitor you have to supply the start address and the number of bytes you want to CSAVE. Or if you use the Dragon's own machine code CSAVE command you use the format:

```
CSAVEM "name",
```

which again is followed by the start address, a comma, the last address to be CSAVEd, another comma, then the entry address. In this case you should not worry too much about the entry address as the machine code routines you have so far will be called by a bootstrap program you'll be given later. For now, though, give any old entry address.

Obviously there is no problem with the start addresses. With machine code assembled from assembly language programming the start address of the object code is the same as the origin. After all, when the assembler is RUN it takes the origin and starts assembling the machine code program from there. And with data tables POKEd in from BASIC, the start address is the initial value of the variable in the FOR . . . NEXT loop which picks up the DATA and stores it in memory a byte at a time.

But working out the number of bytes to be CSAVEd can cause problems. The end address given by assemblers is often the first free address past the end of the program. And sometimes all you are given is the start address of the last instruction, so if the routine does not end with a RTS, it does not give a true end address.

The usual way round this problem is to CSAVE an extra couple of bytes. But that can cause overwriting problems.

The answer is to Load your programs—using the CLOADM command—back into the computer in order up memory. Load the one with the lowest origin or start address first, then the next lowest, then the next and so on. This means that any extra bytes CSAVEd at the end of the routine will be overwritten by the beginning of the next program—instead of the other way round. And it means that any RTS that has been added to test routines will be overwritten too.

When all the routines and data tables have been CLOADed, CSAVE them back to tape or disk as one program under another name—the start address will be the start address of the first program and the end address will be the end address of the last program. Then if you have any problem you can put the various pieces back together and try again. Don't forget to SAVE the assembly language using the SAVE option on your assembler and SAVE the BASIC POKEr programs just in case.

If you have any problems with the data tables, you can LOAD up the machine code routines—in order up memory again—and LOAD up and RUN the BASIC POKEr programs again. These must be LOADed one at a time, RUN and NEWed before you LOAD the next one. Again the whole area of memory should be SAVEd to tape under a new program name. Alternatively you can SAVE the BASIC POKEr program as one large program.

PIECING IT TOGETHER

Putting together a complex graphic image is simplified by assembly-line techniques—'picking and dragging' shapes, and 'rubber-banding' to expand standard elements

Computer Aided Design (CAD) packages can relieve the effort of repetitive design problems such as planning a supermarket, in which a large number of identical objects—shelves, freezers and so on—have to be fitted into a given floor area.

This kind of software ranges from highly sophisticated packages running on large mainframes to simple programs running on home micros, such as *INPUT*'s (see page 566). No matter what the level of sophistication, there are two basic principles employed—the unlikely-sounding 'rubber-banding' and 'picking and dragging'.

This article looks at how you can pick and drag, and rubber-band in BASIC.



RUBBER BANDING

Rubber banding is the name given to the way in which shapes can be drawn using CAD programs.

The idea behind rubber-banding is a simple one—it is easier to draw a straight line by defining the two end points and joining these together with a ruler than it is to draw a line freehand. This idea was used by the early Egyptians in pyramid construction and is still implemented on computers. In the computer version, you set one end of the line, and then allow a cursor to be moved round the screen at the other end. A line is always stretched between the two, and the technique gets its name because it looks like a rubber band as it is stretched and contracted.

Experimenting with various lines on screen will allow you to visualise the alternatives before deciding on the final position. And the same basic idea can be extended from single lines to many, especially those used in geometric and regular shapes.

Try the following program. It will allow you to experiment with rubber-banding for yourself.

```
1Ø BORDER Ø: PAPER Ø: INK 7: CLS
15 LET initialise=8Ø: LET rubberband=
   13Ø: LET draw=21Ø: LET fix=24Ø
17 DRAW Ø,175: DRAW 255,Ø: DRAW
   Ø,−175: DRAW −255,Ø
2Ø GOSUB initialise
4Ø GOSUB rubberband
5Ø IF INKEY$<>"q" THEN GOTO 4Ø
6Ø STOP
9Ø OVER 1
1ØØ LET centrex=128: LET centrey=86
11Ø LET x=128: LET y=86
115 PLOT centrex,centrey
12Ø RETURN
14Ø IF INKEY$="□" THEN GOSUB fix:
   GOTO 15Ø
145 GOSUB draw
15Ø IF INKEY$="z" THEN LET x=x−1
16Ø IF INKEY$="x" THEN LET x=x+1
17Ø IF INKEY$="k" THEN LET y=y+1
18Ø IF INKEY$="m" THEN LET y=y−1
19Ø GOSUB draw
2ØØ RETURN
22Ø PLOT centrex,centrey: DRAW x−PEEK
   23677,y−PEEK 23678
23Ø RETURN
25Ø OVER Ø: GOSUB draw
255 OVER 1
27Ø LET centrex=x: LET centrey=y
28Ø RETURN
```

Like the following Commodore programs, the listing below uses Simon's BASIC to access the high resolution graphics. Commodore users will thus need either to plug in Simons' BASIC cartridge or enter *INPUT*'s own hi-res graphics utility starting on pages 872 to 877. In the latter case, the various hi-res commands need to be prefixed with an @.

```
1Ø HIRES Ø,1:POKE 65Ø,128:
   MULTI Ø,4,5
2Ø CX=8Ø:CY=1ØØ:X=CX:Y=CY
1ØØ GET A$:IF A$="□" THEN LINE
   CX,CY,X,Y,2:CX=X:CY=Y
11Ø IF A$="Z" AND X>1 THEN X=X−2
12Ø IF A$="X" AND X<158
   THEN X=X+2
13Ø IF A$=";" AND Y>1 THEN Y=Y−2
14Ø IF A$="/" AND Y<199
   THEN Y=Y+2
16Ø FOR Z=1 TO 2:LINE CX,CY,X,Y,4:
   NEXT Z
17Ø GOTO 1ØØ
```

- BASIC PRINCIPLES OF COMPUTER AIDED DESIGN
- DRAWING SHAPES BY RUBBER BANDING
- FIXING POINTS
- PICKING AND DRAGGING
- MOVING SHAPES AT WILL
- EXPANDING, CONTRACTING AND ROTATING SHAPES
- FIXING SHAPES

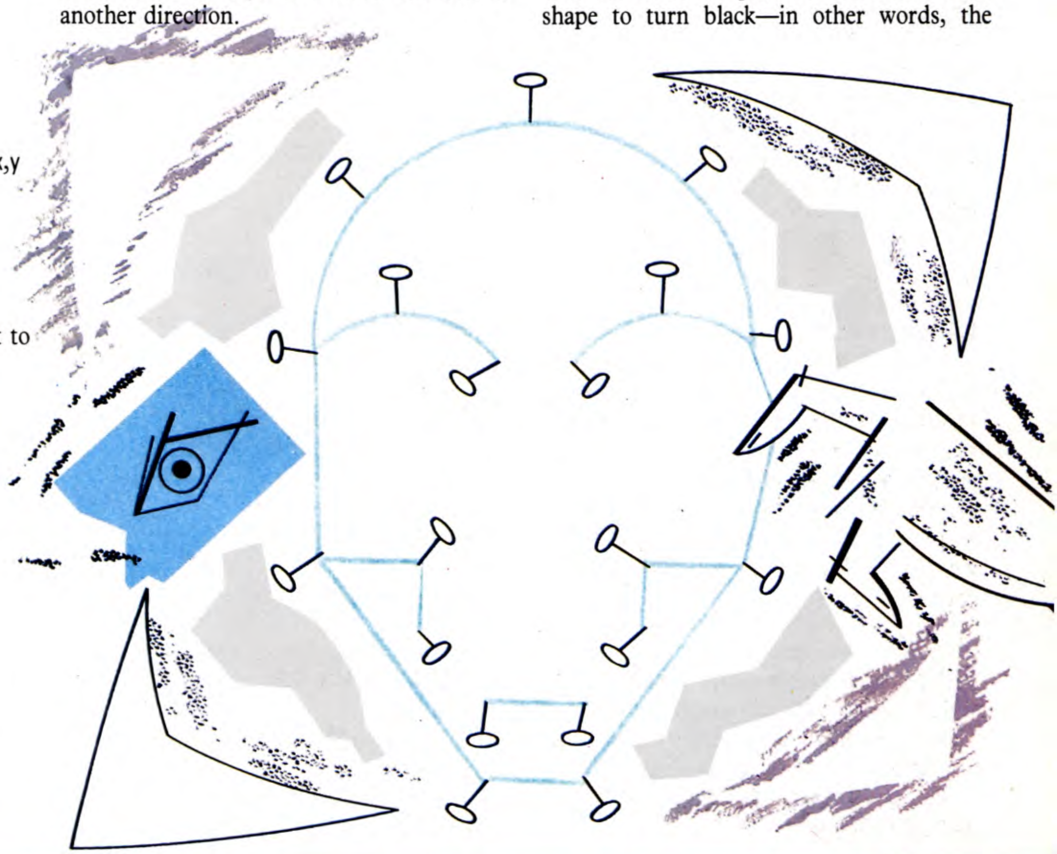
```
1Ø MODE 4
2Ø PROCinitialize
3Ø REPEAT
4Ø PROCrubberband
5Ø UNTIL INKEY(-17)
6Ø MODE 6
7Ø END
8Ø DEF PROCinitialize
9Ø GCOL 3,1
1ØØ centrex=64Ø:centrey=512
11Ø x=64Ø:y=512
12Ø ENDPROC
13Ø DEF PROCrubberband
14Ø PROCdraw
15Ø IF INKEY(-98) THEN x=x-1Ø
16Ø IF INKEY(-67) THEN x=x+1Ø
17Ø IF INKEY(-73) THEN y=y+1Ø
18Ø IF INKEY(-1Ø5) THEN y=y-1Ø
2ØØ ENDPROC
21Ø DEF PROCdraw
217 IFINKEY(-99)THEN PROCfix
22Ø MOVE centrex,centrey:DRAW x,y
225 MOVE centrex,centrey:DRAW x,y
23Ø ENDPROC
24Ø DEF PROCfix
25Ø GCOL Ø,1:MOVE centrex,centrey:DRAWx,y
26Ø GCOL 3,1
27Ø centrex=x:centrey=y
28Ø ENDPROC
```

Tandy owners should ensure that V is set to 247 in Line 2Ø.

```
1Ø PCLEAR8:PMODE4,5:PCLS:
   PMODE4,1:PCLS:SCREEN1,1
2Ø V=223:CX=127:CY=95:
   X=127:Y=95
3Ø GOSUB 1ØØ
4Ø FORK=1TO4:PCOPYK+4TOK:NEXT
5Ø IF PEEK(338)<>191 THEN 3Ø
6Ø CLS:END
1ØØ IF PEEK(345)=V GOSUB 3ØØ
11Ø IF PEEK(341)=V AND Y>Ø THEN
   Y=Y-1
12Ø IF PEEK(342)=V AND Y<191 THEN
   Y=Y+1
13Ø IF PEEK(343)=V AND X>Ø THEN
   X=X-1
14Ø IF PEEK(344)=V AND X<255 THEN
   X=X+1
2ØØ LINE(CX,GY)-(X,Y),PSET
21Ø RETURN
3ØØ GOSUB 2ØØ
31Ø FORK=1TO4:PCOPYK TOK+4:NEXT
32Ø CX=X:CY=Y
33Ø RETURN
```

The programs work very simply. A point in the centre of the screen is fixed, and a second point plotted and replotted at a position determined by four keypresses—Z left, X right, K up and M down (Acorn users should use : for up and / for down, and Dragon, the arrow keys). A line is drawn between the two points continuously, the repeated drawing of the line gives it a flickery appearance.

Once your line has been positioned to your satisfaction, you should press the space bar. This will fix the end point, and you can use the same four keys to stretch the line in another direction.



Acorn users may be a little confused by the use of GCOL in Lines 9Ø and 26Ø. In a two-colour graphics mode, such as Ø or 4, the GCOL 3 statement can be used successfully for animation—moving the line around the screen is really an animation exercise. The parameter 3 instructs the computer to Exclusively OR (EOR) the colour specified with the colour already present on screen.

If you look at this truth table, you'll see how GCOL 3 works. A zero represents the colour specified *not* being present; a one represents the colour specified being present.

Ø EOR Ø gives Ø
Ø EOR 1 gives 1
1 EOR Ø gives 1
1 EOR 1 gives Ø

If the colour specified is white, and the background is not white (black), putting a white shape on the black will give a white shape. Putting a white shape exactly on an identical white shape, will cause the white shape to turn black—in other words, the

shape will disappear. In this program, then, the continual drawing and redrawing of the line turns it on and then off at each position, giving the animation effect.

Now add these lines and see how much more flexible rubber-banding becomes when you can erase lines.

```
15 LET initialize=8Ø: LET rubberband=
   13Ø: LET draw=21Ø: LET fix=24Ø: LET
   clear=31Ø
17 GOSUB clear
144 IF INKEY$="c"THEN GOSUB
   clear:GOTO15Ø
31Ø CLS
32Ø DRAW Ø,175: DRAW 255,Ø: DRAW
   Ø,-175: DRAW -255,Ø
33Ø GOSUB initialize
34Ø RETURN
```

```
1Ø5 IF A$=CHR$(13) THEN
   CX=X:CY=Y
```

```
135 *FX 15,1
145 IF INKEY(-83) THEN PROCclear
215 IF INKEY(-51) THEN
   GCOLØ,Ø:GOTO225 ELSE GCOL3,1
3ØØ DEF PROCclear
31Ø CLG
35Ø PROCinitialize
36Ø ENDPROC
```

```
1Ø5 IF PEEK(339)=191 THEN
   PMODE4,5:PCLS:PMODE4,1
3Ø1 A$=INKEY$
3Ø2 A$=INKEY$:IF A$="" THEN 3Ø2
3Ø4 IF A$="D" THEN 32Ø
```

The Commodore and Dragon/Tandy programs simply draw out the line when the D key is pressed (c in Spectrum), but again the Acorn program uses GCOL to erase the line. In Line 215, GCOL Ø,Ø draws a black line over whatever is displayed on screen.

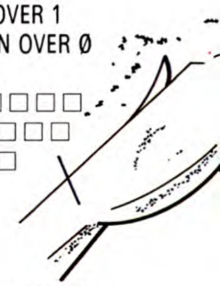
PICKING AND DRAGGING

In many CAD packages there is a range of predefined shapes presented as a menu. In an application such as room design, these shapes would be the various fittings and furniture, but picking and dragging, as it is called, can equally be applied to printed circuit design, where the predefined shapes will be the various electronic components that are to be used in the circuit.

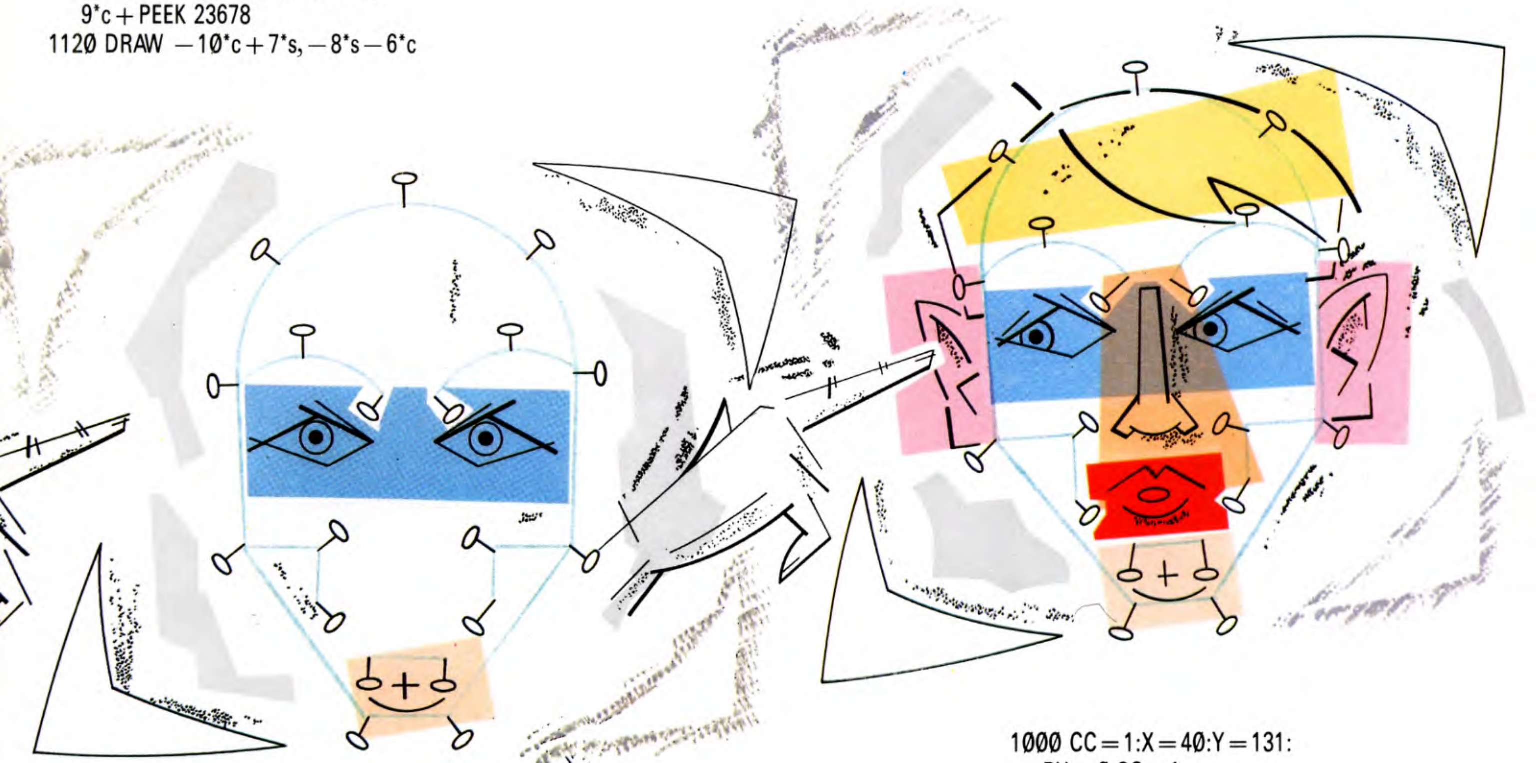
But why is this called picking and dragging? Simply because to use this kind of package, you must pick a shape, and drag it to whatever position you wish.

The program that follows allows you to pick and drag from a range of five geometrical shapes. In addition, you can plot them in any position on screen, and go on to manipulate another shape; expand or contract a shape; rotate a shape; or erase the screen.

```
1Ø BORDER Ø: PAPER Ø: INK 7: OVER Ø: CLS
15 LET initialize=1ØØ: LET pick=27Ø: LET
   give=43Ø: LET drag=49Ø: LET
   point=39Ø: LET draw=64Ø: LET
   fix=7ØØ: LET clear=75Ø: LET
   check=82Ø: LET triang=86Ø: LET
   square=93Ø: LET rectan=1Ø1Ø: LET
   pent=1Ø9Ø: LET hex=118Ø
17 GOSUB clear
4Ø GOSUB pick
5Ø GOSUB give
6Ø GOSUB drag
7Ø IF INKEY$<>"q" THEN GOTO 4Ø
8Ø STOP
11Ø OVER Ø
12Ø LET x=3Ø: LET y=5Ø: LET phi=Ø: LET
   scal=1
13Ø LET mex=12Ø: LET flag=Ø: LET
   select=Ø
14Ø LET c=scal*COS (phi)
15Ø LET s=scal*SIN (phi)
17Ø LET tx=32: LET ty=25: GOSUB triang
19Ø LET tx=7Ø: LET ty=25: GOSUB square
21Ø LET tx=12Ø: LET ty=25: GOSUB rectan
22Ø LET tx=18Ø: LET ty=25: GOSUB pent
23Ø LET tx=23Ø: LET ty=25: GOSUB hex
24Ø OVER 1
25Ø PRINT OVER 1; INK 6;AT 2Ø,mex/8;"∧"
26Ø RETURN
3ØØ GOSUB point
31Ø GOSUB check
32Ø IF CODE INKEY$=8 THEN LET
   mex=mex-8
33Ø IF CODE INKEY$=9 THEN LET
   mex=mex+8
34Ø GOSUB point
35Ø IF INKEY$="s" THEN LET flag=1
36Ø IF flag=Ø THEN GOTO 3ØØ
37Ø GOSUB point
38Ø RETURN
4ØØ PRINT INK 6; OVER 1;AT 2Ø,mex/8;"∧"
41Ø RETURN
44Ø IF mex<4Ø THEN LET select=1
45Ø IF mex>=4Ø AND mex<7Ø THEN LET
   select=2
46Ø IF mex>=7Ø AND mex<15Ø THEN LET
   select=3
465 IF mex>=15Ø AND mex<18Ø THEN
   LET select=4
47Ø IF mex>=18Ø THEN LET
   select=5
48Ø RETURN
51Ø IF INKEY$="c" THEN GOSUB clear:
   GOTO 517
515 GOSUB draw
517 IF INKEY$="□" THEN GOSUB fix:
   GOTO 52Ø
518 GOSUB draw
52Ø LET c=scal*COS (phi)
54Ø LET s=scal*SIN (phi)
55Ø IF INKEY$="z" THEN LET x=x-3
56Ø IF INKEY$="x" THEN LET x=x+3
57Ø IF INKEY$="k" THEN LET y=y+3
58Ø IF INKEY$="m" THEN LET y=y-3
59Ø IF INKEY$="n" THEN LET scal=scal*1.1
6ØØ IF INKEY$="j" THEN LET scal=scal/1.1
61Ø IF INKEY$="h" THEN LET
   phi=phi+(6*PI/18Ø)
62Ø IF INKEY$="b" THEN LET
   phi=phi-(6*PI/18Ø)
63Ø RETURN
645 LET tx=x: LET ty=y
65Ø IF select=1 THEN GOSUB triang
66Ø IF select=2 THEN GOSUB square
67Ø IF select=3 THEN GOSUB rectan
675 IF select=4 THEN GOSUB pent
68Ø IF select=5 THEN GOSUB hex
685 FOR n=1 TO 3Ø: NEXT n
69Ø RETURN
7ØØ REM fix
71Ø FOR n=1 TO 1ØØ: NEXT n
714 IF INKEY$="d" THEN OVER 1
715 IF INKEY$<>"d" THEN OVER Ø
717 GOSUB draw
72Ø PRINT AT 2Ø,1;"□□□□□□
   □□□□□□□□□□□□□□
   □□□□□□□□□□□□
   □□□□": OVER 1
73Ø GOSUB initialize
74Ø RETURN
76Ø CLS
77Ø DRAW Ø,175: DRAW 255,Ø: DRAW
   Ø,-175: DRAW -255,Ø
78Ø GOSUB initialize
79Ø RETURN
82Ø REM check
83Ø IF INKEY$="c" THEN GOSUB clear
84Ø IF INKEY$="e" THEN STOP
85Ø RETURN
86Ø REM triang
87Ø PLOT INVERSE 1;tx,ty
88Ø PLOT -7*s+PEEK 23677,6*c+PEEK
   23678
89Ø DRAW -6*c+11*s,-5*s-9*c
9ØØ DRAW 12*c,1Ø*s
91Ø DRAW -6*c-11*s,-5*s+9*c
92Ø RETURN
93Ø REM square
94Ø PLOT INVERSE 1;tx,ty
95Ø PLOT 6*c-6*s+PEEK 23677,
   5*s+5*c+PEEK 23678
96Ø DRAW -12*c,-1Ø*s
```



```
970 DRAW 12*s,-1Ø*c
980 DRAW 12*c,1Ø*s
990 DRAW -12*s,1Ø*c
1ØØØ RETURN
1Ø1Ø REM rectan
1Ø2Ø PLOT INVERSE 1;tx,ty
1Ø3Ø PLOT 12*c-6*s+PEEK 23677,
   1Ø*s+5*c+PEEK 23678
1Ø4Ø DRAW -24*c,-2Ø*s
1Ø5Ø DRAW 12*s,-1Ø*c
1Ø6Ø DRAW 24*c,2Ø*s
1Ø7Ø DRAW -12*s,1Ø*c
1Ø8Ø RETURN
1Ø9Ø REM pent
11ØØ PLOT INVERSE 1;tx,ty
111Ø PLOT -1Ø*s+PEEK 23677,
   9*c+PEEK 23678
112Ø DRAW -1Ø*c+7*s,-8*s-6*c
113Ø DRAW 4*c+13*s,3*s-11*c
114Ø DRAW 12*c,1Ø*s
115Ø DRAW 4*c-13*s,3*s+11*c
116Ø DRAW -1Ø*c-7*s,-8*s+6*c
117Ø RETURN
118Ø REM hex
119Ø PLOT INVERSE 1;tx,ty
12ØØ PLOT -12*s+PEEK 23677,
   1Ø*c+PEEK 23678
121Ø DRAW -1Ø*c+6*s,-8*s-5*c
122Ø DRAW 12*s,-1Ø*c
123Ø DRAW 1Ø*c+6*s,8*s-5*c
124Ø DRAW 1Ø*c-6*s,8*s+5*c
125Ø DRAW -12*s,1Ø*c
126Ø DRAW -1Ø*c-6*s,-8*s+5*c
127Ø RETURN
```



The program works as follows:

Lines 11Ø to 26Ø are the initialization subroutine, setting values for a range of variables, and drawing the shapes along the bottom of the screen by calling the appropriate subroutines.

Lines 3ØØ to 38Ø read the left and right cursor keys, allowing you to move the arrow, and the s key, which makes the picked shape appear at the centre of the screen. Lines 4ØØ and 41Ø are the point subroutine, replotting the arrow in response to the cursor key presses.

Lines 44Ø to 48Ø are the give subroutine. It checks the position of the arrow in relation to the range of shapes. The select variable is set according to the position of the pointer.

The drag subroutine is at Lines 51Ø to 63Ø. The selected shape is redrawn in response to key presses. The shape can be moved around using the z, x, k and m keys; n and j allow you to scale the shape up or down; and h and b allow you to rotate the shape clockwise or anticlockwise. The space bar fixes the shape on screen.

The shapes are drawn by a combination of the draw subroutine, between Lines 645 and 69Ø, and the various shapes subroutines—**select** tells the program which one is to be drawn.

The fix subroutine—Lines 71Ø to 74Ø—fixes the shape on screen in response to the d key. Lines 76Ø to 79Ø are the clear subroutine, which is called when the c key is pressed—the check is in the check subroutine. The check subroutine—Lines 82Ø to 84Ø—also checks for the escape option—pressing the e key—which stops the program.

```
1Ø HIRES Ø,1:POKE 65Ø,128
2Ø GOSUB 1ØØØ
3Ø GOSUB 2ØØØ
4Ø GOSUB 3ØØØ
5Ø GOSUB 4ØØØ
6Ø GOTO 2Ø
1ØØØ CC=1:X=4Ø:Y=131:
   PH=Ø:SC=1
1Ø1Ø ME=122
1Ø2Ø C=SC:S=Ø
1Ø3Ø X=6Ø:Y=177:GOSUB 5ØØØ
1Ø4Ø X=1ØØ:Y=174:GOSUB 51ØØ
1Ø5Ø X=14Ø:GOSUB 52ØØ
1Ø6Ø X=18Ø:Y=17Ø:GOSUB 53ØØ
1Ø7Ø X=22Ø:GOSUB 54ØØ
1Ø9Ø X=16Ø:Y=1ØØ
11ØØ RETURN
2ØØØ GOSUB 25ØØ
2Ø1Ø GET A$
2Ø2Ø IF A$="▮▮" AND ME>52 THEN
   ME=ME-1Ø
2Ø3Ø IF A$="▮" AND ME<232 THEN
   ME=ME+1Ø
2Ø5Ø IF A$<>"S" THEN 2ØØØ
2Ø7Ø RETURN
25ØØ FOR Z=1 TO 2:TEXT ME,19Ø,
```

```
   " ✝ ",2,1,8:NEXT Z
251Ø RETURN
3ØØØ SL=5:IF ME<222 THEN SL=4
3Ø1Ø IF ME<172 THEN SL=3
3Ø2Ø IF ME<112 THEN SL=2
3Ø3Ø IF ME<82 THEN SL=1
3Ø4Ø RETURN
4ØØØ CC=2:FOR Z=1 TO 2:ON SL GOSUB
   5ØØØ,51ØØ,52ØØ,53ØØ,
   54ØØ:NEXT Z
4Ø1Ø GET A$
4Ø2Ø IF A$="□" THEN CC=1:ON SL
   GOSUB 5ØØØ,51ØØ,52ØØ,53ØØ,
   54ØØ:RETURN
4Ø3Ø C=SC*COS(PH)
4Ø4Ø S=SC*SIN(PH)
4Ø5Ø IF A$="▌" THEN X=X-3
4Ø6Ø IF A$="▟" THEN X=X+3
4Ø7Ø IF A$="◘" THEN Y=Y-3
4Ø8Ø IF A$="◙" THEN Y=Y+3
4Ø9Ø IF A$="M" THEN SC=SC*1.1
41ØØ IF A$="N" THEN SC=SC/1.1
411Ø IF A$="K" THEN
   PH=PH+ATN(1)/7.5
412Ø IF A$="L" THEN
   PH=PH-ATN(1)/7.5
414Ø IF A$=CHR$(13) THEN
   RETURN
4145 IF A$="Q" THEN END
415Ø GOTO 4ØØØ
5ØØØ LINE X-7.2*S,Y-7.2*C,
   X-6*C+3.6*S,Y+6*S+3.6*C,CC
5Ø1Ø LINE X-6*C+3.6*S,Y+6*S+3.6*C,
   X+6*C+3.6*S,Y-6*S+3.6*C,CC
5Ø2Ø LINE X+6*C+3.6*S,
   Y-6*S+3.6*C,X-7.2*S,Y-7.2*C,CC
5Ø3Ø RETURN
51ØØ LINE X+6*S-6*C,Y-6*S-6*C,
   X-6*S-6*C,Y+6*S-6*C,CC
511Ø LINE X-6*S-6*C,Y+6*S-6*C,
   X+6*S-6*C,Y+6*S+6*C,CC
512Ø LINE X+6*S-6*C,Y+6*S+6*C,
   X+6*S+6*C,Y+6*C-6*S,CC
513Ø LINE X+6*S+6*C,Y+6*C-6*S,
   X+6*C-6*S,Y-6*S-6*C,CC
514Ø RETURN
52ØØ LINE X+12*C-6*C,Y-6*C-12*S,
   X-12*C-6*S,Y-6*C+12*S,CC
521Ø LINE X-12*C-6*S,Y-6*C+12*S,
   X-12*C+6*S,Y+6*C+12*S,CC
522Ø LINE X-12*C+6*S,Y+6*C+12*S,
   X+12*C+6*S,Y+6*C-12*S,CC
523Ø LINE X+12*C+6*S,Y+6*C-12*S,
   X+12*C-6*S,Y-6*C-12*S,CC
524Ø RETURN
53ØØ LINE X-1Ø.2*S,Y-1Ø.2*C,
   X-9.8*C-3.2*S,Y-3.2*C+9.8*S,CC
531Ø LINE X-9.8*C-3.2*S,
   Y-3.2*C+9.8*S,X-6*C+1Ø.2*S,
   Y+1Ø.2*C+6*S,CC
532Ø LINE X-6*C+1Ø.2*S,Y+1Ø.2*C+6*S,
   X+6*C+1Ø.2*S,Y+1Ø.2*C-6*S,CC
533Ø LINE X+6*C+1Ø.2*S,Y+1Ø.2*C-6*S,
   X+9.8*C-3.2*S,Y-3.2*C-9.8*S,CC
534Ø LINE X+9.8*C-3.2*S,
   Y-3.2*C-9.8*S,X-1Ø.2*S,Y-1Ø.2*C,
   CC
535Ø RETURN
54ØØ LINE X-12*S,Y-12*C,
   X-1Ø.4*C-6*S,Y-6*C+1Ø.4*S,CC
541Ø LINE X-1Ø.4*C-6*S,Y-6*C+1Ø.4*S,
   X-1Ø.4*C+6*S,Y+6*C+1Ø.4*S,CC
542Ø LINE X-1Ø.4*C+6*S,Y+6*C+1Ø.4*S,
   X+12*S,Y+12*C,CC
543Ø LINE X+12*S,Y+12*C,
   X+1Ø.4*C+6*S,Y+6*C-1Ø.4*S,CC
544Ø LINE X+1Ø.4*C+6*S,Y+6*C-1Ø.4*S,
   X+1Ø.4*C-6*S,Y-6*C-1Ø.4*S,CC
545Ø LINE X+1Ø.4*C-6*S,Y-6*C-1Ø.4*S,
   X-12*S,Y-12*C,CC
546Ø RETURN
```



Tandy owners should alter the values of V1 and V2 in Line 1Ø1Ø. Make V1 equal to 247, and V2 equal to 253.

```
1Ø PCLEAR8:PMODE4,5:PCLS:
   PMODE4,1:PCLS:SCREEN1,1
2Ø GOSUB 1ØØØ
4Ø GOSUB 2ØØØ
5Ø GOSUB 3ØØØ
6Ø GOSUB 4ØØØ
7Ø IF INKEY$<>"Q" THEN 4Ø
8Ø CLS:END
1ØØØ X=2Ø:Y=131:PH=Ø:SC=1
1Ø1Ø ME=122:ST=Ø:V1=223:
   V2=247
1Ø2Ø C=SC:S=Ø:COLOR5
1Ø3Ø X=4Ø:Y=177:GOSUB5ØØØ
1Ø4Ø X=8Ø:Y=174:GOSUB51ØØ
1Ø5Ø X=12Ø:GOSUB52ØØ
1Ø6Ø X=16Ø:Y=17Ø:GOSUB53ØØ
1Ø7Ø X=2ØØ:GOSUB54ØØ
1Ø8Ø FORK=1TO4:PCOPYK TOK+4:NEXT
1Ø9Ø X=127:Y=85
11ØØ RETURN
2ØØØ COLORØ:GOSUB 25ØØ
2Ø1Ø A$=INKEY$:GOSUB45ØØ
2Ø2Ø IF A$=CHR$(8) AND ME>32 THEN
   ME=ME-1Ø
2Ø3Ø IF A$=CHR$(9) AND ME<212 THEN
   ME=ME+1Ø
2Ø4Ø COLOR5:GOSUB25ØØ
2Ø5Ø IF A$<>"S" THEN 2ØØØ
2Ø6Ø COLOR5:GOSUB25ØØ
2Ø7Ø RETURN
25ØØ DRAW"BM"+STR$(ME)+
   ",19ØU5NG2F2"
251Ø RETURN
3ØØØ SL=5:IF ME<192 THEN SL=4
3Ø1Ø IF ME<152 THEN SL=3
3Ø2Ø IF ME<1Ø2 THEN SL=2
3Ø3Ø IF ME<62 THEN SL=1
3Ø4Ø RETURN
4ØØØ ON SL GOSUB 5ØØØ,51ØØ,52ØØ,
   53ØØ,54ØØ
4Ø1Ø IF PEEK(339)=191 THEN
```

```
    PCLS:GOSUB1ØØØ:RETURN
4Ø2Ø IF PEEK(345)=V1 THEN FOR
    K=1TO4:PCOPYK TOK+4:NEXT:
    RETURN
4Ø3Ø C=SC*COS(PH)
4Ø4Ø S=SC*SIN(PH)
4Ø5Ø IF PEEK(343)=VI THEN X=X−1
4Ø6Ø IF PEEK(344)=V1 THEN X=X+1
4Ø7Ø IF PEEK(341)=V1 THEN Y=Y−1
4Ø8Ø IF PEEK(342)=V1 THEN Y=Y+1
4Ø9Ø IF PEEK(343)=V2 THEN SC=SC*1.1
41ØØ IF PEEK(344)=V2 THEN SC=SC/1.1
411Ø IF PEEK(341)=V2 THEN
    PH=PH+ATN(1)/7.5
412Ø IF PEEK(342)=V2 THEN
    PH=PH−ATN(1)/7.5
413Ø FORK=1TO4:PCOPYK+4TOK:NEXT
415Ø IF PEEK(338)=V1 THEN RETURN
    ELSE4ØØØ
45ØØ IF A$=CHR$(12) THEN PCLS:
    GOSUB1ØØØ
451Ø IF A$="Q" THENCLS:END
452Ø RETURN
5ØØØ LINE(X−7.2*S,Y−7.2*C)−
    (X−6*C+3.6*S,Y+6*S+3.6*C),PSET
5Ø1Ø LINE−(X+6*C+3.6*S,Y−6*S+
    3.6*C),PSET
5Ø2Ø LINE−(X−7.2*S,Y−7.2*C),PSET
5Ø3Ø RETURN
51ØØ LINE(X+6*C−6*S,Y−6*S−6*C)−
    (X−6*S−6*C,Y+6*S−6*C),PSET
511Ø LINE−(X+6*S−6*C,Y+6*S+6*C),
    PSET
512Ø LINE−(X+6*S+6*C,Y+6*C−6*S),
    PSET
513Ø LINE−(X+6*C−6*S,Y−6*S−6*C),
    PSET
514Ø RETURN
52ØØ LINE(X+12*C−6*S,Y−6*C−12*S)−
    (X−12*C−6*S,Y−6*C+12*S),PSET
521Ø LINE−(X−12*C+6*S,Y+6*C+12*S),
    PSET
522Ø LINE−(X+12*C+6*S,Y+6*C−12*S),
    PSET
523Ø LINE−(X+12*C−6*S,Y−6*C−12*S),
    PSET
524Ø RETURN
53ØØ LINE(X−1Ø.2*S,Y−1Ø.2*C)−
    (X−9.8*C−3.2*S,Y−3.2*C+9.8*S),PSET
531Ø LINE−(X−6*C+1Ø.2*S,Y+1Ø.2*C+
    6*S),PSET
532Ø LINE−(X+6*C+1Ø.2*S,Y+1Ø.2*C−
    6*S),PSET
533Ø LINE−(X+9.8*C−3.2*S,Y−3.2*C−
    9.8*S),PSET
534Ø LINE−(X−1Ø.2*S,Y−1Ø.2*C),PSET
535Ø RETURN
54ØØ LINE(X−12*S,Y−12*C)−
    (X−1Ø.4*C−6*S,Y−6*C+1Ø.4*S),PSET
541Ø LINE−(X−1Ø.4*C+6*S,Y+6*C+
    1Ø.4*S),PSET
542Ø LINE−(X+12*S,Y+12*C),PSET
543Ø LINE−(X+1Ø.4*C+6*S,Y+6*C−
    1Ø.4*S),PSET
544Ø LINE−(X+1Ø.4*C−6*S,Y−6*C−
    1Ø.4*S),PSET
545Ø LINE−(X−12*S,Y−12*C),PSET
546Ø RETURN
```

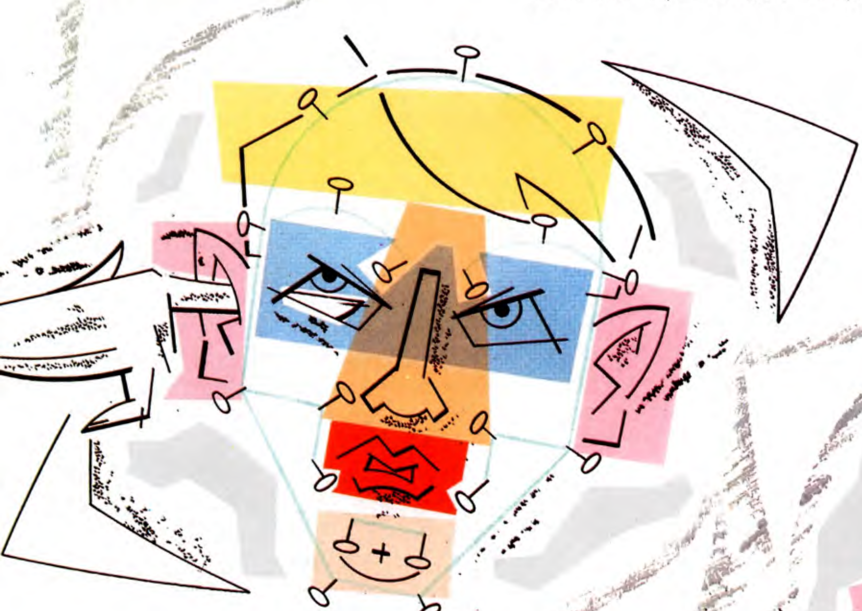
Initialization takes place between Lines 1ØØØ and 11ØØ. Lines 2ØØØ to 2Ø7Ø are the picking subroutine. The arrow beneath the range of five shapes is moved in response to the left and right cursor controls.

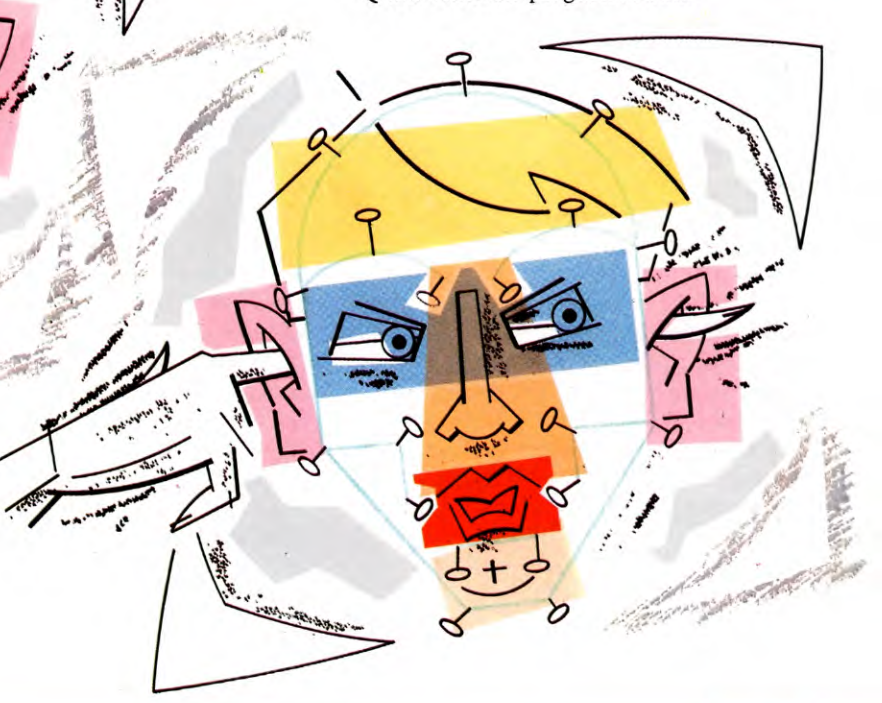
Lines 3ØØØ to 3Ø4Ø determine which shape is being pointed at—ME is the position of the arrow. Pressing the S key will make the shape appear in the main screen area.

Lines 4ØØØ to 415Ø are the drawing subroutine. Line 4ØØØ directs the program to the subroutine containing the instructions for drawing the chosen shape. Line 4Ø2Ø fixes the shape on screen if the space bar is pressed. Lines 4Ø9Ø and 41ØØ read the M and N keys, which cause the shape to get bigger or smaller. The final option is rotation—Lines 411Ø and 412Ø read the K and L keys. Pressing CLEAR will clear the screen, ready for next effort.

The remaining subroutines contain instructions for drawing the five shapes. Lines 5ØØØ to 5Ø3Ø draw the triangle; Lines 51ØØ to 514Ø draw the square; Lines 52ØØ to 524Ø draw the rectangle; Lines 53ØØ to 535Ø draw the pentagon, and Lines 54ØØ to 546Ø draw the hexagon.

There is no error checking in the program, so any attempt to draw off the screen to the top or left will produce an FC ERROR. Pressing Q will cause the program to end.





```
1Ø MODE4
2Ø PROCinitialize
3Ø REPEAT
4Ø PROCpick
5Ø PROCgive
6Ø PROCdrag
7Ø UNTIL INKEY(−17)
8Ø MODE 6
9Ø END
1ØØ DEF PROCinitialize
11Ø GCOL Ø,1
12Ø x=1ØØ:y=3ØØ:phi=Ø:scal=1
13Ø mex=6Ø8:flag=FALSE:select=Ø
14Ø c=scal*COS(phi)
15Ø s=scal*SIN(phi)
16Ø VDU 5
17ØPROCtriang(12Ø,1ØØ)
18Ø MOVE Ø,2ØØ:PRINT;"triangle"
19Ø PROCsquare(36Ø,1ØØ)
2ØØ MOVE 272,2ØØ:PRINT;"square"
21Ø PROCrectan(62Ø,1ØØ)
22Ø MOVE 482,2ØØ:PRINT;"rectangle"
23Ø PROChexgon(88Ø,1ØØ)
24Ø MOVE 782,2ØØ:PRINT;"hexagon"
25Ø PROCpentgon(113Ø,1ØØ)
26Ø MOVE 1Ø14,2ØØ:PRINT"pentagon"
27Ø GCOL 3,1
28Ø ENDPROC
29Ø DEF PROCpick
3ØØ PROCpoint(mex)
31Ø REPEAT
32Ø PROCpoint(mex)
33Ø PROCcheck
34Ø IF INKEY(−26)
   THEN mex=mex−1Ø
35Ø IF INKEY(−122)
   THENmex=mex+1Ø
36Ø PROCpoint(mex)
37Ø IF INKEY(−82) THENflag=TRUE
38Ø UNTIL flag
39Ø PROCpoint(mex)
4ØØ ENDPROC
41Ø DEF PROCpoint(menux)
42Ø MOVE menux,2Ø
43Ø VDU 94
44Ø ENDPROC
45Ø DEF PROCgive
46Ø IF mex<2ØØ THEN select=1
47Ø IF mex>=2ØØ AND mex<45Ø THEN
select=2
48Ø IF mex>=45Ø AND mex<72Ø THEN
   select=3
49Ø IF mex>=72Ø AND mex<97Ø THEN
   select=4
5ØØ IF mex>97Ø THEN select=5
51Ø ENDPROC
52Ø DEF PROCdrag
54Ø IF INKEY(−83) THEN PROCclear ELSE
   PROCdraw(select)
55Ø IF INKEY(−99) THEN PROCfix ELSE
   PROCdraw(select)
56Ø c=scal*COS(phi)
57Ø s=scal*SIN(phi)
58Ø IF INKEY(−98)
   THEN x=x−1Ø
59Ø IF INKEY(−67)
   THEN x=x+1Ø
6ØØ IF INKEY(−73)
   THEN y=y+1Ø
61Ø IF INKEY(−1Ø5)
   THEN y=y−1Ø
62Ø IF INKEY(−1Ø2)
   THEN scal=scal*1.1
63Ø IF INKEY(−86)
   THEN scal=scal/1.1
64Ø IF INKEY(−71)
   THEN phi=phi+RAD(6)
65Ø IF INKEY(−87)
   THEN phi=phi−RAD(6)
66Ø ENDPROC
67Ø DEF PROCdraw(select)
68Ø IF select=1
   THEN PROCtriang(x,y)
69Ø IF select=2
   THEN PROCsquare(x,y)
7ØØ IF select=3
   THEN PROCrectan(x,y)
71Ø IF select=4 THEN PROChexgon(x,y)
72Ø IF select=5 THEN PROCpentgon(x,y)
73Ø ENDPROC
74Ø DEF PROCfix
75Ø GCOL Ø,1:PROCdraw(select)
76Ø GCOL 3,1
77Ø PROCinitialise
78Ø ENDPROC
79Ø DEF PROCclear
8ØØ GCOL Ø,Ø
81Ø MOVE Ø,21Ø:MOVE 1279,21Ø
82Ø PLOT 85,1279,1Ø23:MOVE Ø,1Ø23
83Ø PLOT 85,Ø,21Ø
84Ø PROCinitialise
85Ø ENDPROC
86Ø DEF PROCcheck
87Ø IF INKEY(−83) THEN PROCclear
88Ø IF INKEY(−17) THEN END
89Ø ENDPROC
9ØØ DEF PROCtriang(x,y)
91Ø MOVE x,y
92Ø PLOT Ø,−36*s,36*c
93Ø PLOT 1,−3Ø*c+54*s,−3Ø*s−54*c
94Ø PLOT 1,6Ø*c,6Ø*s
95Ø PLOT 1,−3Ø*c−54*s,−3Ø*s+54*c
96Ø ENDPROC
97Ø DEF PROCsquare(x,y)
98Ø MOVE x,y
99Ø PLOT Ø,3Ø*c−3Ø*s,3Ø*s+3Ø*c
1ØØØ PLOT 1,−6Ø*c,−6Ø*s
1Ø1Ø PLOT 1,6Ø*s,−6Ø*c
1Ø2Ø PLOT 1,6Ø*c,6Ø*s
1Ø3Ø PLOT1,−6Ø*s,6Ø*c
1Ø4Ø ENDPROC
1Ø5Ø DEF PROCrectan(x,y)
1Ø6Ø MOVE x,y
1Ø7Ø PLOT Ø, 6Ø*c−3Ø*s,6Ø*s+3Ø*c
1Ø8Ø PLOT 1,−12Ø*c,−12Ø*s
1Ø9Ø PLOT 1,6Ø*s,−6Ø*c
11ØØ PLOT1, 12Ø*c,12Ø*s
111Ø PLOT 1,−6Ø*s,6Ø*c
112Ø ENDPROC
113Ø DEF PROCpentgon(x,y)
114Ø MOVE x,y
115Ø PLOT Ø,−51*s,51*c
116Ø PLOT 1,−49*c+35*s,−49*s−35*c
117Ø PLOT 1,19*c+67*s,19*s−67*c
118Ø PLOT 1,6Ø*c,6Ø*s
119Ø PLOT 1,19*c−67*s,19*s+67*c
12ØØ PLOT 1,−49*c−35*s,−49*s+35*c
121Ø ENDPROC
122Ø DEF PROChexgon(x,y)
123Ø MOVE x,y
124Ø PLOT Ø,−6Ø*s,6Ø*c
125Ø PLOT 1,−52*c+3Ø*s,−52*s−3Ø*c
126Ø PLOT 1,6Ø*s,−6Ø*c
127Ø PLOT 1,52*c+3Ø*s,52*s−3Ø*c
128Ø PLOT 1,52*c−3Ø*s,52*s+3Ø*c
129Ø PLOT 1,−6Ø*s,6Ø*c
13ØØ PLOT 1,−52*c−3Ø*s,−52*s+3Ø*c
131Ø ENDPROC
```

The program works like this:

After initialization, the program enters the program's main loop—the REPEAT...UNTIL between Lines 3Ø and 7Ø. The program will run until the Q key is pressed, which will END the program. PROCinitialize sets the size, position and orientation of the shape you choose. The five shapes are drawn at the bottom of the screen.

PROCpick, between Lines 29Ø and 4ØØ, reads the left and right cursor keys, so you can point to the desired shape. PROCpoint is to be found between Lines 41Ø and 43Ø, and prints the arrow on screen.

PROCgive checks the position of the arrow, and sets select according to the shape being pointed at. PROCdrag first checks for a press on the C key—this clears the screen—and then checks for a press on the space bar—this fixes the shape on the screen. The rest of the PROCedure reads the Z,X,: and / keys, which move the shape, and then read the M and N keys, which alter the size of the shape, and the K and L keys which rotate the shape.

PROCdraw calls the appropriate shape drawing PROCedure, according to the value of select which has been passed. The shapes are drawn by the PROCedures at the end of the program, and their functions are given by their names.

Finally, PROCcheck reads the C and Q keys. A press on C will clear the screen, and Q will quit the program.

CONTROLLING THE BOARD-2

- PART TWO OF OTHELLO
- COMPLETING THE PLAYER'S TURN
- THE COMPUTER'S MOVE
- ANNOUNCING THE WINNER

After the interval, *INPUT*'s Othello game comes to its climax. At the end of the second half you'll have the complete game to act upon. Now make up some wily moves . . .

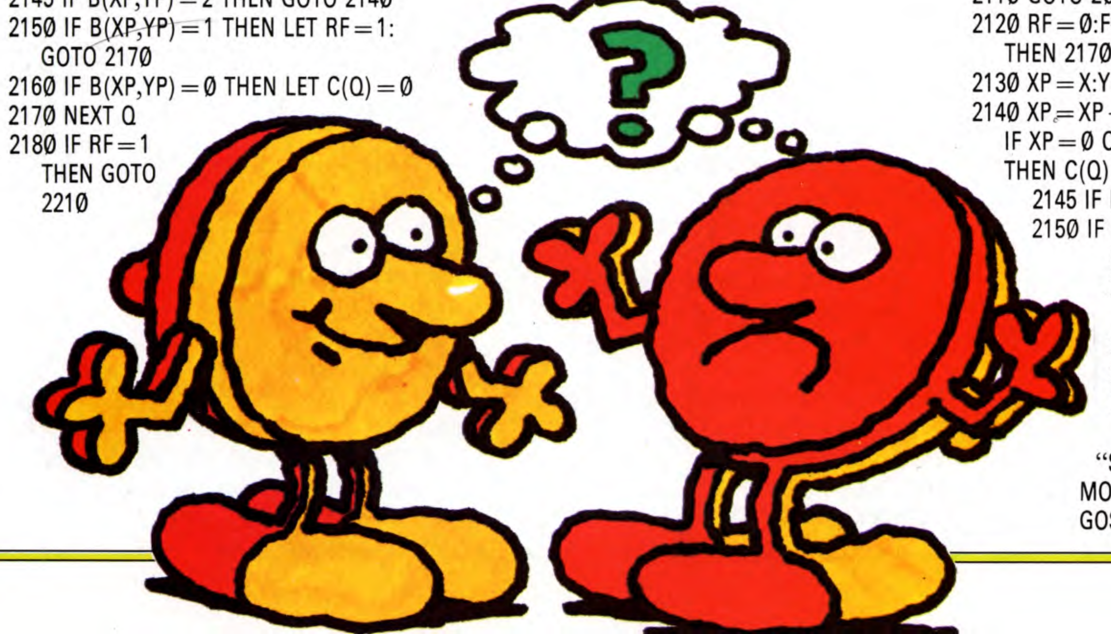
When you have completed this second part of the Othello game you will have the whole program to pit your wits against. This time, here are the lines needed to complete the player's turn, along with the computer's move routine, and the end of game routine.

AFTER THE INTERVAL

```
2090 IF NF=1 THEN GOTO 2120
2100 PRINT AT 17,0;"YOUR MOVE ISN'T
   NEXT TO ONE OF□□MY PIECES": FOR
   F=1 TO 500: NEXT F
2110 PRINT AT 17,0;"□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□": GOTO 2000
2120 LET RF=0: FOR Q=1 TO 8: IF
   C(Q)=0 THEN GOTO 2170
2130 LET XP=X: LET YP=Y
2140 LET XP=XP+D(Q,1): LET
   YP=YP+D(Q,2): IF XP=0 OR XP=9
   OR YP=0 OR YP=9 THEN LET C(Q)=0:
   GOTO 2170
2145 IF B(XP,YP)=2 THEN GOTO 2140
2150 IF B(XP,YP)=1 THEN LET RF=1:
   GOTO 2170
2160 IF B(XP,YP)=0 THEN LET C(Q)=0
2170 NEXT Q
2180 IF RF=1
   THEN GOTO
   2210
2190 PRINT AT 17,0;"YOUR MOVE DOES NOT
   FLANK A ROW": FOR F=1 TO 500: NEXT
   F
2200 PRINT AT 17,0;"□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□□□□□□": GOTO 2000
2210 FOR Q=1 TO 8: IF C(Q)=0 THEN
   GOTO 2250
2220 LET XP=X+D(Q,1): LET
   YP=Y+D(Q,2)
2230 IF B(XP,YP)=1 THEN GOTO 2250
2240 LET B(XP,YP)=1: LET
   XP=XP+D(Q,1): LET YP=YP+D(Q,2):
   GOTO 2230
2250 NEXT Q
2260 LET B(X,Y)=1
2270 LET CP=2: RETURN
```

```
2090 IFNF=1THEN2120
2100 PRINT"■YOUR MOVE ISN'T
   NEXT TO ONE OF MY PIECES":
   FORF=0TO1500:NEXT
2110 PRINT"◙◙□□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□◙◙◙◙◙":GOTO2000
2120 RF=0:FORQ=1TO8:IFC(Q)=0
   THEN2170
2130 XP=X:YP=Y
2140 XP=XP+D(Q,1):YP=YP+
   D(Q,2):IF(XP=0ORXP=9)OR
   (YP=0ORYP=9)THENC(Q)=0:
   GOTO2170
2145 IFB(XP,YP)=2THEN2140
2150 IFB(XP,YP)=1THENRF=1:
   GOTO2170
2160 IFB(XP,YP)=0THENC(Q)=0
2170 NEXT
2180 IFRF=1THEN2210
2190 PRINT"■YOUR MOVE DOES
   NOT FLANK A ROW":FORF=0TO
   1500:NEXT
2200 PRINT"◙□□□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□□□□□□◙◙◙◙":
   GOTO2000
2210 FORQ=1TO8:IFC(Q)=0THEN2250
2220 XP=X+D(Q,1):YP=Y+D(Q,2)
2230 IFB(XP,YP)=1THEN2250
2240 B(XP,YP)=1:XP=XP+D(Q,1):
   YP=YP+D(Q,2):GOTO2230
2250 NEXTQ
2260 B(X,Y)=1
2270 CP=2:RETURN
```

```
2090 IF NF=1 THEN 2120
2100 COLOR1:LINE(0,182)-(255,
   191),PSET,BF:DRAW"C4S8BM0,
   182":A$="THAT ISNT NEXT TO ONE OF
   MINE":GOSUB 9300: FOR F=1 TO
   900:NEXTF
2110 GOTO 2000
2120 RF=0:FOR Q=1 TO 8:IF C(Q)=0
   THEN 2170
2130 XP=X:YP=Y
2140 XP=XP+D(Q,1):YP=YP+D(Q,2):
   IF XP=0 OR XP=9 OR YP=0 OR YP=9
   THEN C(Q)=0:GOTO 2170
2145 IF B(XP,YP)=2 THEN 2140
2150 IF B(XP,YP)=1 THEN RF=1:
   GOTO 2170
2160 IF B(XP,YP)=0 THEN
   C(Q)=0
2170 NEXT Q
2180 IF RF=1 THEN 2210
2190 COLOR1:LINE (0,182)-
   (255, 191),PSET,BF:DRAW
   "S8C4BM0, 182":A$="YOUR
   MOVE DOESNT FLANK A ROW":
   GOSUB 9300:FOR F=1 TO
```



```
    900:NEXTF
2200 GOTO 2000
2210 FOR Q=1 TO 8:IF C(Q)=0
    THEN 2250
2220 XP=X+D(Q,1):YP=Y+D(Q,2)
2230 IF B(XP,YP)=1 THEN 2250
2240 B(XP,YP)=1:XP=XP+D(Q,1):
    YP=YP+D(Q,2):GOTO 2230
2250 NEXT Q
2260 B(X,Y)=1
2270 CP=2:RETURN
```

Line 2Ø9Ø checks the flag NF, to make sure that there is a piece belonging to the computer in an adjoining square before jumping to Line 212Ø. If there isn't a piece, Line 21ØØ displays an error message.

The program next checks in Lines 212Ø to 217Ø to see if the move flanks a row.

Line 214Ø makes sure that the position being checked falls within the boundaries of the board. If it doesn't, then that direction is abandoned and the next tried. Line 2145 makes sure that the piece being checked does belong to the computer. If it does, then the program jumps back to Line 214Ø to update the position.

In Line 218Ø the success of the last operation is checked. If a row was found the program jumps to Line 221Ø. Lines 219Ø to 22ØØ PRINT a message to tell the player that a row has not been flanked and the program jumps back to Line 2ØØØ.

The move itself is made in Lines 221Ø to 226Ø. The loop in Line 221Ø checks if a suitable row has been found by looking at C(Q). If there is not a row in that direction the program jumps to the NEXT Q at Line 225Ø. XP and YP are set to the first square of the row being taken—see Line 2Ø4Ø.

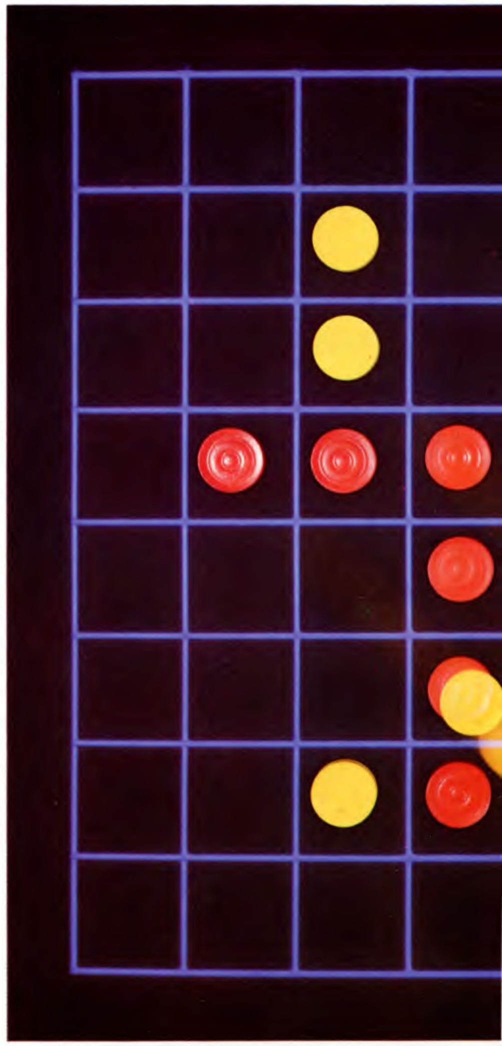
In Line 223Ø the computer checks for the end of a row. If an end is found, the program jumps to Line 225Ø. Line 224Ø sets the square to one then jumps back to Line 223Ø to check the next square.

In Line 226Ø the player's square is set to one. The CP flag is set to two for the computer in Line 227Ø and the program RETURNS.

THE COMPUTER'S MOVE

```
3000 PRINT '"THINKING ...": LET NF=1:
    LET MX=0: FOR X=1 TO 8: FOR Y=1
    TO 8
3010 IF B(X,Y)<>0 THEN GOTO 3070
3020 FOR F=1 TO 8: LET XP=X: LET
    YP=Y: LET DX=D(F,1): LET DY=D(F,2):
    LET RF=0
3030 LET XP=XP+DY: LET YP=YP+DX: IF
    XP=0 OR XP=9 OR YP=0 OR YP=9
    THEN GOTO 3060
3040 IF B(XP,YP)=1 THEN LET RF=1:
    GOTO 3030
3050 IF B(XP,YP)=2 AND RF=1 THEN LET
    N(NF)=F: LET X(NF)=X: LET Y(NF)=Y:
    LET NF=NF+1: LET F=9
3060 NEXT F
3070 NEXT Y: NEXT X: LET NF=NF-1
3075 IF NF=0 THEN GOTO 3300
3080 FOR F=1 TO NF: LET X=X(F): LET
    Y=Y(F): LET DX=D(N(F),1): LET
    DY=D(N(F),2): LET CF=0
3090 LET X=X+DY: LET Y=Y+DX: IF
    B(X,Y)=1 THEN LET CF=CF+1: GOTO
    3090
3100 IF CF>MX THEN LET MX=CF: LET
    MF=F
3110 NEXT F
3180 FOR F=1 TO 8: LET X=X(MF): LET
    Y=Y(MF): LET DX=D(F,1): LET
    DY=D(F,2)
3190 LET X=X+DY: LET Y=Y+DX
3195 IF X<1 OR X>8 OR Y<1 OR Y>8
    THEN GOTO 3260
3200 IF B(X,Y)=1 THEN GOTO 3190
3210 IF B(X,Y)=2 THEN GOTO 3230
3220 IF B(X,Y)=0 THEN GOTO 3260
3230 LET X=X(MF): LET Y=Y(MF)
3235 LET B(X,Y)=2: LET X=X+DY: LET
    Y=Y+DX
3240 IF B(X,Y)=2 THEN GOTO 3260
3250 GOTO 3235
3260 NEXT F
3265 PRINT X(MF),Y(MF): INPUT A$
3270 LET CP=1: RETURN
3300 PRINTAT 17,0;"I CANNOT
    MAKE A MOVE":FOR F=1 TO
    500:NEXT F
3305 PRINTAT 17,0;"□□□□□□
    □□□□□□□□□□□□□□
    □□□□□□□□□□□□□□"
3310 LET CP=1
3320 RETURN
```

```
3000 NF=1:MX=0:FORX=1TO8:
    FORY=1TO8
3010 IFB(X,Y)<>0THEN3070
3020 FORF=1TO8:XP=X:YP=Y:
    DX=D(F,1):DY=D(F,2):RF=0
3030 XP=XP+DY:YP=YP+DX:IF
    (XP=0ORXP=9)OR(YP=0ORYP=9)
    THEN3060
3040 IFB(XP,YP)=1THENRF=1:
    GOTO3030
3050 IFB(XP,YP)=2AND RF=1 THEN
    N(NF)=F:X(NF)=X:Y(NF)=Y:
    NF=NF+1:F=9
3060 NEXT F
3070 NEXT Y:NEXT X:NF=NF-1
```



```
3075 IF NF=0 THEN 3300
3080 FORF=1TONF:X=X(F):Y=
    Y(F):DX=D(N(F),1):DY=D(N
    (F),2):CF=0
3090 X=X+DY:Y=Y+DX:IFB(X,Y)
    =1THENCF=CF+1:GOTO3090
3100 IFCF>MXTHENMX=CF:MF=F
3110 NEXT
3180 FORF=1TO8:X=X(MF):Y=Y
    (MF):DX=D(F,1):DY=D(F,2)
3190 X=X+DY:Y=Y+DX
3195 IF X<1 ORX>8 OR Y<1 OR Y>8
    THEN3260
3200 IFB(X,Y)=1THEN3190
3210 IFB(X,Y)=2THEN3230
3220 IFB(X,Y)=0THEN3260
3230 X=X(MF):Y=Y(MF)
3235 B(X,Y)=2:X=X+DY:Y=Y+DX
3240 IFB(X,Y)=2THEN3260
3250 GOTO3235
3260 NEXT
3265 PRINTX(MF),Y(MF):POKE
    198,0:WAIT198,1:POKE 198,0
3270 CP=1:RETURN
```



```
3300 PRINT "▣ICANNOT MAKE A
   MOVE":FOR F=1 TO1500:NEXT
3310 PRINT "◘←□□□□□□□□□□
   □□□□□□□□□□:CP=1: RETURN
```

```
3000 COLOR1:LINE (0,182)-(255,191),
   PSET,BF:A$="COMPUTERS MOVE":
   DRAW "S8C3BM66,182":GOSUB 9300:
   NF=1:MX=0:FOR X=1 TO 8:FOR Y=1
   TO 8
3010 IF B(X,Y)<>0 THEN 3070
3020 FOR F=1 TO 8:XP=X:YP=Y:
   DX=D(F,1):DY=D(F,2):RF=0
3030 XP=XP+DY:YP=YP+DX:IF XP=0
   OR XP=9 OR YP=0 OR YP=9 THEN
   3060
3040 IF B(XP,YP)=1 THEN RF=1:GOTO
   3030
3050 IF B(XP,YP)=2 AND RF=1
   THEN N(NF)=F:X(NF)=X:
   Y(NF)=Y:NF=NF+1:F=9
3060 NEXT F
3070 NEXT Y,X:NF=NF-1
3075 IF NF=0THEN 3300
3080 FOR F=1 TO NF:X=X(F):Y=Y
   (F):DX=D(N(F),1):DY=D(N(F),2):
   CF=0
3090 X=X+DY:Y=Y+DX:IF B(X,Y)=1
   THEN CF=CF+1:GOTO 3090
3100 IF CF>MX THEN MX=CF:MF=F
3110 NEXT F
3180 FOR F=1 TO 8:X=X(MF):Y=Y
   (MF):DX=D(F,1):DY=D(F,2)
3190 X=X+DY:Y=Y+DX
3195 IF X<1 OR X>8 OR Y<1 OR Y>8
   THEN 3260
3200 IF B(X,Y)=1 THEN 3190
3210 IF B(X,Y)=2 THEN 3230
3220 IF B(X,Y)=0 THEN 3260
3230 X=X(MF):Y=Y(MF)
3235 B(X,Y)=2:X=X+DY:Y=Y+DX
3240 IF B(X,Y)=2 THEN 3260
3250 GOTO 3235
3260 NEXT F
3265 DRAW "S16;C2;BM118,150"+NU$
   (X(MF))+NU$(Y(MF))+"S8"
3270 CP=1:RETURN
3300 COLOR1:LINE(0,182)-(255,191),
   PSET,BF:A$="I CANNOT MOVE":DRAW
   "S8C4BM66,182":GOSUB9300:FORF=1
   TO 50:NEXT:CP=1:RETURN
```

The number of squares in a row is set to one, and the maximum number of pieces found in a row is set to zero in Line 3000. Two loops—with control variables X and Y—are initiated. These hunt through the board for empty spaces. It is this section of program which is really time-consuming. As the number of empty squares decreases as the game progresses, so the time taken for the computer's move also decreases.

Line 3010 checks to see if the square is empty. If it isn't, then the computer jumps to the NEXT statements in Line 3070. Lines 3020 to 3060 check to see if the square is at the end of a row which the computer can take. XP and YP are used as before. DX and DY represent the direction array D() contents and save quite a lot of space.

Line 3030 checks to see if the square to be tested lies on the board. If it doesn't, the next direction is tested. If the square tested is a player's square then the routine jumps back to Line 3030 to check if the next square is occupied by the player.

In Line 3050 the board is checked to see if it is occupied by the computer. If it is and a row has been found, the start positions are recorded in X() and Y(), the direction number being stored in N(). The counter to indicate the number of coordinates found is also increased.

Only the first row is stored, to ensure ensure that the search takes as little time as possible.



Lines 3080 to 3110 find which move gives the longest score in a straight line. A loop from one to NF (number of squares found) is set up. X and Y are equated with X(F) and Y(F). The direction coordinates, DX and DY, are set to the directions indicated by N(F). CF, a temporary count is zeroed on each execution of the loop.

Line 3090 checks to see if the piece being tested is a player's piece. If it is, then CF is increased and the next square in that direction tested. Line 3100 tests to see if the number found (CF) is greater than the previous maximum (MX). If it is, then MX is set to equal CF, and MF, the coordinates of the 'best' piece set to the loop index F.

The move routine is carried out by Lines 3180 to 3260. Lines 3180 starts a loop from one to eight. This is done so that rows in all directions can be found.

X, Y, DX and DY are set as before, and in Line 3195, X and Y are tested to ensure they are still on the board. If they are not, the program jumps to the NEXT at Line 3260.

Line 3200 checks to see if the player occupies this square. If he does, the routine jumps to try the next square in the line. At this point no squares are altered, the routine is only testing.

If the row ends in a computer occupied square, X and Y are reset and Lines 3235 to 3250 alter all the squares in the row. If an empty square is found, the next direction is tried.

Once the routine has decided on the square it wishes to move into, Line 3265 PRINTs the coordinates and waits for a key to be pressed (specifically, <ENTER> on the Spectrum). Line 3270 sets the CP flag to one for the player's turn, then RETURNs to the main loop.

END OF GAME

```
4ØØØ IF PS>CS THEN GOTO 5ØØØ
4Ø1Ø IF PS=CS THEN GOTO 6ØØØ
4Ø2Ø PRINT AT 17,Ø; INK 2;"THAT WAS EASY!"
4Ø3Ø PRINT "DO YOU WANT ANOTHER
    GAME ('Y' OR 'N')"
4Ø4Ø LET A$=INKEY$: IF A$<>"Y" AND
    A$<>"N" THEN GOTO 4Ø4Ø
4Ø5Ø IF A$="Y" THEN RUN
4Ø6Ø STOP
5ØØØ PRINT AT 17,Ø; INK 2;"YOU WERE
    LUCKY!"
5Ø1Ø GOTO 4Ø3Ø
6ØØØ PRINT AT 17,Ø; INK 2;"WE DREW,I
    NEED MORE PRACTICE": GOTO 4Ø3Ø
```

```
4ØØØ IFPS>CSTHEN5ØØØ
4Ø1Ø IFPS=CSTHEN6ØØØ
4Ø2Ø PRINT"THAT WAS EASY!"
4Ø3Ø PRINT"DO YOU WANT ANOTHER
    GAME('Y' OR 'N')"
4Ø4Ø GETA$:IFA$<>"Y"ANDA$<>
    "N"THEN4Ø4Ø
4Ø5Ø IFA$="Y"THENRUN
4Ø6Ø END
5ØØØ PRINT"YOU WERE LUCKY!"
5Ø1Ø GOTO4Ø3Ø
6ØØØ PRINT"WE DREW,I NEED MORE
    PRACTICE":GOTO4Ø3Ø
```

```
4ØØØ IF PS>CS THEN 5ØØØ
4Ø1Ø IF PS=CS THEN 6ØØØ
4Ø15 A$="THAT WAS EASY"
4Ø2Ø COLOR1:LINE (Ø,182)–(255,191),
    PSET,BF:DRAW "S8C2BM7Ø,182":
    GOSUB 93ØØ
4Ø25 FOR F=1 TO 15ØØ:NEXTF
4Ø3Ø COLOR1:LINE (Ø,182)–(255,191),
    PSET,BF:A$=" DO YOU WANT ANOTHER
    GAME":DRAW "C3BMØ,182":
    GOSUB 93ØØ
4Ø4Ø A$=INKEY$:IF A$<>"Y" AND
    A$<>"N" THEN 4Ø4Ø
4Ø5Ø IF A$="Y" THEN RUN
4Ø6Ø END
5ØØØ A$="YOU WERE LUCKY"
5Ø1Ø GOTO 4Ø2Ø
6ØØØ A$=" IT IS A DRAW":GOTO 4Ø2Ø
```

The end of game routine is located from Line 4ØØØ onwards. Lines 4ØØØ itself checks to see if the player has won by comparing PS and CS. The program jumps to Line 5ØØØ to PRINT the win message. Line 4Ø1Ø tests for a draw, and the message routine is located at Line 6ØØØ. If the computer has won, the program reaches Line 4Ø2Ø and displays a message to rub salt in the wound!

The remaining lines are simply an 'another go?' option.

Again, the Acorn program has been written differently from the others, so the routines are a little different from those described earlier.

YOUR MOVE

```
44Ø DEFPROChumanmove
45Ø *FX15,1
46Ø COLOUR 2:COLOUR 128:INPUT TAB
    (Ø,25)SPC(8Ø)TAB(Ø,25),"What
    is your move?""(row,col) ", y%,x%
47Ø IF x%=Ø AND y%=Ø THEN
    eg%=1:ENDPROC
475 IF y%=9 THEN cp%=2:ENDPROC
48Ø IF x%<1 OR x%>8 OR y%<1 OR
    y%>8 THEN VDU 7:GOTO 45Ø
49Ø IF b%(x%,y%)<>Ø THEN
    PROCbadmove(1):GOTO 45Ø
5ØØ nf%=Ø:FOR f%=1 TO 8: cf%=Ø:IF
    x%+d%(f%,1)=Ø OR x%+d%(f%,1)=9
    THEN 53Ø
51Ø IF y%+d%(f%,2)=Ø OR
    y%+d%(f%,2)=9 THEN 53Ø
52Ø IF b%(x%+d%(f%,1),y%+
    d%(f%,2))=2 THEN cf%=1: nf%=1
53Ø c%(f%)=Ø:IF cf%=1 THEN
    c%(f%)=f%
54Ø NEXT f%
55Ø IF nf%<>1 THEN PROCbad
    move(2):GOTO 45Ø
56Ø rf%=Ø:FOR q%=1 TO 8:IF c%(q%)=Ø
    THEN 63Ø
57Ø xp%=x%:LET yp%=y%
58Ø xp%=xp%+d%(q%,1):
    yp%=yp%+d%(q%,2)
59Ø IF xp%=Ø OR xp%=9 OR yp%=Ø OR
    yp%=9 THEN c%(q%)=Ø:GOTO 63Ø
6ØØ IF b%(xp%,yp%)=2 THEN 58Ø
61Ø IF b%(xp%,yp%)=1 THEN rf%=1:GOTO
    63Ø
62Ø IF b%(xp%,yp%)=Ø THEN c%(q%)=Ø
63Ø NEXT q%
64Ø IF rf%<>1 THEN PROCbadmove
    (3):GOTO 45Ø
65Ø FOR q%=1 TO 8:IF c%(q%)=Ø THEN
    71Ø
66Ø xp%=x%+d%(q%,1):
    yp%=y%+d%(q%,2)
67Ø IF b%(xp%,yp%)=1 THEN 71Ø
68Ø b%(xp%,yp%)=1
69Ø xp%=xp%+d%(q%,1):
    yp%=yp%+d%(q%,2)
7ØØ GOTO 67Ø
71Ø NEXT q%
72Ø b%(x%,y%)=1: cp%=2
73Ø PROCdisplayboard
74Ø ENDPROC
```

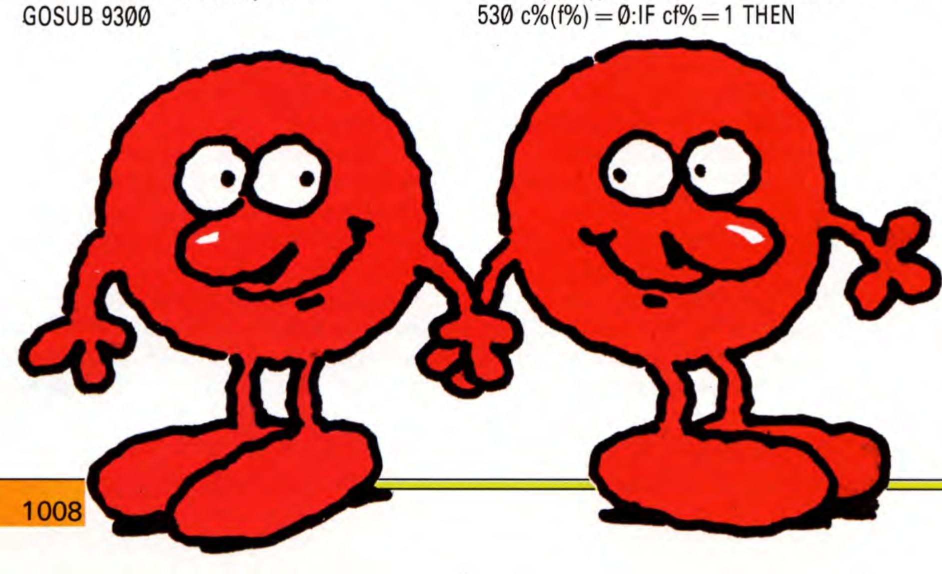
The player enters a move in response to the prompt. Line 47Ø checks if the player wants to stop the game, and Line 48Ø checks if the input is outside the correct range.

If the square is already occupied, Line 49Ø calls PROCbadmove. Lines 5ØØ to 55Ø check if the move is into a square that is next to one of the computer's pieces. If it isn't, then Line 55Ø calls PROCbadmove. Lines 56Ø to 64Ø check if the move is into a position that flanks a row. If it doesn't, then Line 64Ø calls PROCbadmove once again. Each of these lines passes a different parameter value to PROCbadmove which determines the message that appears on screen.

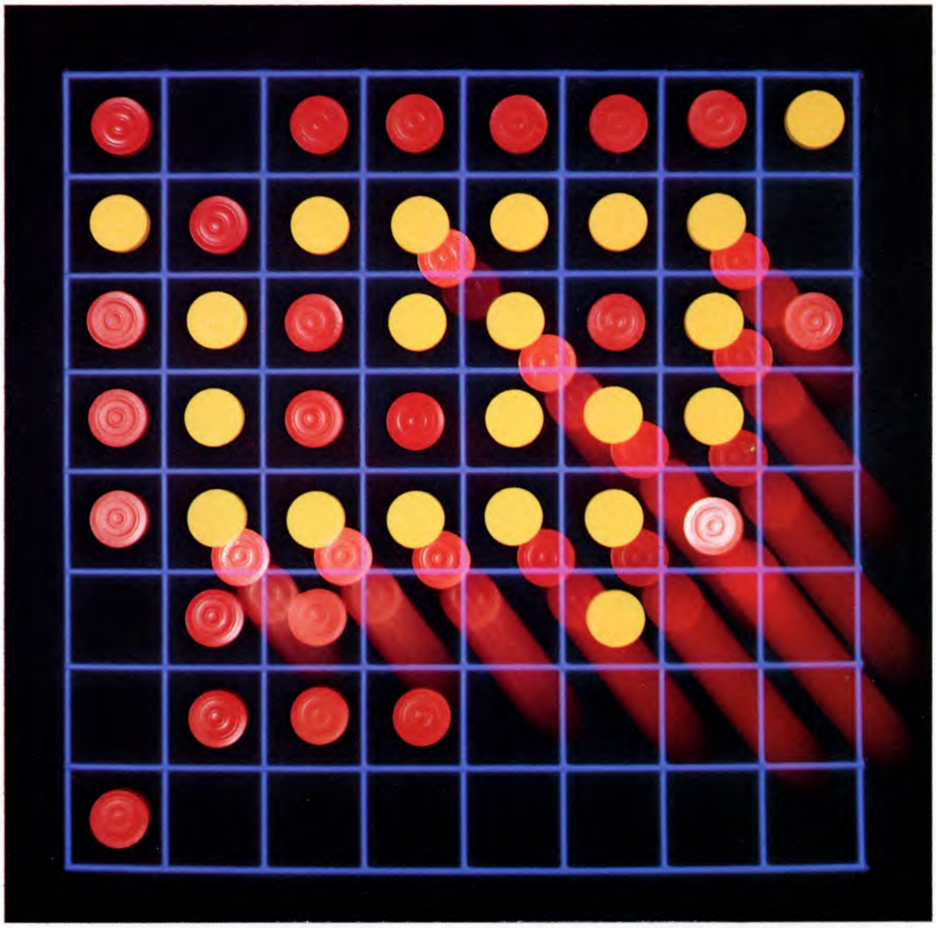
The final section of the PROCedure sets the arrays ready for PROCdisplay board to display the new status of the board—pieces will always change status after a move has been made.

COMPUTER'S MOVE

```
75Ø DEFPROCcomputermove
76Ø COLOUR 3: PRINT TAB(Ø,25);"Thinking
    ..."SPC 4Ø
77Ø nf%=1: mx%=Ø
78Ø FOR x%=1 TO 8
79Ø FOR y%=1 TO 8
8ØØ IF b%(x%,y%)<>Ø THEN 89Ø
```



```
81Ø FOR f%=1 TO 8
82Ø xp%=x%: yp%=y%
83Ø dx%=d%(f%,1): dy%=d%(f%,2):
    rf%=Ø
84Ø xp%=xp%+dy%: yp%=yp%+dx%
85Ø IF xp%=Ø OR xp%=9 OR yp%=Ø OR
    yp%=9 THEN 88Ø
86Ø IF b%(xp%,yp%)=1 THEN rf%=1:GOTO
    84Ø
87Ø IF b%(xp%,yp%)=2 AND rf%=1 THEN
    n%(nf%)=f%: x%(nf%)=x%:
    y%(nf%)=y%: nf%=nf%+1:LET f%=9
88Ø NEXT f%
89Ø NEXT y%
9ØØ NEXT x%:LET nf%=nf%-1
905 IF nf%=Ø THEN12Ø5
91Ø FOR f%=1 TO nf%
92Ø x%=x%(f%)
93Ø y%=y%(f%)
94Ø dx%=d%(n%(f%),1)
95Ø dy%=d%(n%(f%),2): cf%=Ø
96Ø REPEAT
97Ø x%=x%+dy%: y%=y%+dx%
98Ø IF b%(x%,y%)=1 THEN cf%=cf%+1
99Ø UNTIL b%(x%,y%)<>1
1ØØØ IF cf%>mx% THEN mx%=cf%:
     mf%=f%
1Ø1Ø NEXT f%
1Ø2Ø FOR f%=1 TO 8
1Ø3Ø x%=x%(mf%): y%=y%(mf%)
1Ø4Ø dx%=d%(f%,1): dy%=d%(f%,2)
1Ø5Ø x%=x%+dy%:
     y%=y%+dx%
1Ø6Ø IF x%<1 OR x%>8 OR y%<1 OR
     y%>8 THEN 116Ø
1Ø7Ø IF b%(x%,y%)=1
     THEN 1Ø5Ø
1Ø8Ø IF b%(x%,y%)=2
     THEN 11ØØ
1Ø9Ø IF b%(x%,y%)=Ø
     THEN 116Ø
11ØØ x%=x%(mf%):LET y%=y%(mf%)
111Ø REPEAT
112Ø b%(x%,y%)=2
113Ø x%=x%+dy%
114Ø y%=y%+dx%
115Ø UNTIL b%(x%,y%)=2
116Ø NEXT f%
117Ø PRINT TAB(Ø,25);"My move is□";
     y%(mf%);",";x%(mf%):*FX15
118Øa%=INKEY(1ØØØ)
119Ø cp%=1
12ØØ ENDPROC
12Ø5 PRINT"I CANNOT MAKE A MOVE":
     FOR F=1 TO 2ØØØ: NEXT:ENDPROC
```



PROCcomputer move follows the same general structure as PROChuman's move, but must also choose a move for the computer.

In Line 78Ø the number of squares in a row is set to 1, and the maximum number of pieces found in a row is set to zero. This is the computer's start point for its explorations for a good move. Two loops are initiated in Lines 78Ø and 79Ø. These hunt through the board for empty spaces.

It is this section of program which is really time-consuming. As the number of empty squares decreases as the game progresses, so the time taken for the computer's move also decreases. The computer is searching for empty squares which are at the end of rows of pieces.

Lines 1ØØØ to 116Ø look for the best possible position for the new piece. Line 117Ø displays the move. You are given ten seconds to look at the move before it is plotted. The wait can be overridden by tapping any key.

BAD MOVES

```
121Ø DEFPROCbadmove(t%)
122Ø COLOUR 1:VDU 7:PRINT TAB(Ø,25);
123Ø IF t%=1 THEN PRINT "You cannot move
     ontoan occupied square."
124Ø IF t%=2 THEN PRINT "Your move isn't
     nextto one of my pieces."
125Ø IF t%=3 THEN PRINT "Your move
     doesn't□□□flank a row.□□□"
126Ø *FX15,1
127Ø a%=INKEY(5ØØ)
128Ø PRINT TAB(Ø,25);SPC(4Ø)
129Ø ENDPROC
```

PROCbad move contains three error messages. Parameters are passed from PROChuman's move if bad moves are found. t% is set according to the nature of the bad moves, and Lines 123Ø and 125Ø display the message.

THE END

```
13ØØ DEFPROCgameover
131Ø COLOUR 3:PRINT TAB(Ø,25);
     SPC(8Ø);TAB(Ø,25);
132Ø IF ps%>cs% THEN PRINT"You were
     lucky!"
133Ø IF ps%<cs% THEN PRINT"That was
     easy!"
134Ø IF ps%=cs% THEN PRINT"It was a
     draw!"
135Ø REPEAT
136Ø COLOUR 2:PRINTTAB(Ø,3Ø);
     "Play again (Y/N) ?"
137Ø x$=GET$
138Ø UNTIL INSTR("yYnN",x$)>Ø
139Ø IF x$="y" OR x$="Y" THEN RUN
     ELSE CLS:COLOUR 3:END
14ØØ ENDPROC
```

LET'S MAKE A DATE

Plan out all the important events in the coming year and keep a note of special dates such as birthdays with this comprehensive calendar and diary program.



Are you the sort of person who always forgets your mother's birthday or only remembers a dental appointment a day too late? And are you surprised when a bill drops through the letterbox even though you *know* it appears regularly every quarter?

The program that accompanies these articles will keep track of all these things for you. It is easy to use and much more fun than writing entries into a diary, so you'll have no excuse to miss an appointment or forget to pay the rates ever again. If you have a printer you can make a hard copy of the diary to carry around with you so you can check what's coming even when you are away from the computer.

AUTOMATIC CALENDAR

The program really does two things, printing out either a calendar or a diary. The simplest option prints out a calendar for any month in any year between 1753 and 29,999 (or 3299 on the Acorns). The calendar is displayed in the usual way with the days of the week along the top and the numbers of the days underneath. The program automatically takes account of leap years, and the date of Easter Sunday is printed out underneath the month in which it falls.

As well as looking at the single monthly calendar you can also choose to print out a calendar for a whole year. This is useful if you have a printer as you can keep the calendar by your desk or pin it on the wall, and there is plenty of room on the printout to make notes by any of the dates.

ELECTRONIC DIARY

The diary option is the one that lets you keep track of what's happening. The entries are made under four separate headings—Finance, Appointments, Celebrations and Holidays—and each is highlighted in a different colour so when you use the diary it is easy to pick out a particular type of entry.

Entering the information is very straightforward, and this program has an advantage over an ordinary diary in that it will automatically carry forward regular events such as bills or birthdays, filling them in on the correct date for all following months and years. For example, when you make an entry under Finance you are first asked whether it is monthly, quarterly, annual or just a single one-off event. Say you're entering details of your monthly payments. You would type M for monthly then enter the name RENT followed by the date you first paid. The word RENT will then appear on that date for every following month.

The Celebrations option is for things such as birthdays and anniversaries so these are automatically taken as annual events. The Appointments and Holidays are all treated as single events.

It doesn't matter in what order you enter the dates, the computer will sort through them all and group them together under the correct month.

Once you've entered the data, it is best to save it straight away, using the SAVE option in

- MONTHLY CALENDAR
- CALCULATING EASTER
- YEARLY CALENDAR
- FILLING IN THE DIARY
- ENTERING REGULAR EVENTS
- FINANCE, APPOINTMENTS, CELEBRATIONS AND HOLIDAYS
- HOW TO USE THE DIARY
- ENTERING THE FIRST PART OF THE PROGRAM





the menu, in case any disasters happen with the program. The data is stored in a separate file called DIARY and can be loaded in, amended, updated or deleted at any time. So you can easily keep your diary up to date.

USING THE DIARY

Once you've entered some data into the program you can see how the monthly diary works. Enter the number of the month and the year and you'll see all the entries for that month, including any regular finances or celebrations carried forward from earlier months or years. The entries are grouped together in the different categories, each highlighted in the same colours as those used for the monthly calendar.

The program waits for you to press a key before each group is printed on the screen. This is in case there are too many entries to fit on the screen in one go, and so it stops early entries being scrolled off the screen before you've had a chance to read them. If you have a printer attached to the computer you can choose to have a printout of all the entries.

If you now go back to the calendar option and print out a calendar for the same month, you can highlight the dates corresponding to the diary entries by pressing F, A, C or H for each of the four categories. Again, they are colour-coded so you know which category the dates refer to, although you need the diary to see the details of each entry.

The best way to use the program depends on exactly what you want to find out. If you want to see what's coming in a particular month then select the month diary. If, instead, you want to scan through a year to see, for example, how many financial events to plan for, then the quickest way is to select the monthly calendar for the first month, highlight the financial dates by pressing F, move on to the next month and so on for the rest of the year. Once you use the program you'll see just how versatile it is.

ENTERING THE PROGRAM

The program is rather long and so has been split into three parts. The first part given here sets up several of the routines and prints out the main menu. The remainder of the program follows later and there will also be more detailed instructions on how to use it. There are also changes to make to the Acorn program so it can run on the Electron. Don't forget to SAVE this part of the program so you can add the extra routines next time.

```
10 DATA 2,4,1,3,7,31,28,31,30,31,30,31,
   31,30,31,30,31
20 DATA "MNLY","QRLY","ANLY","SNGL"
30 BORDER 0: PAPER 0: INK 7: CLS
40 CLS
50 CLEAR : LET P=2
60 LET Z$="□□□□□□□□□□
   □□□□□□□□□□□□□□□□
   □□□□□□□□□"
70 DIM O$(1,31): DIM Q(4): DIM
   L$(4,150,31): DIM T$(4,12): DIM C(4):
   DIM Z(5)
80 FOR n=1 TO 5: READ Z(n): NEXT n
90 DIM D(12): FOR n=1 TO 12: READ D(n):
   NEXT n
100 LET M$="January□□February□March
   □□□□April□□□□May□□□
   □□□June□□□□□July□□□
   □□August□□□SeptemberOctober□□
   November□December□"
110 LET s$="SunMonTueWedThuFriSat"
120 DIM P$(4,4): FOR n=1 TO 4: READ
   P$(n): NEXT n
130 LET T$(1)="Finances": LET T$(2)=
   "Appointments": LET T$(3)=
   "Celebrations": LET T$(4)= "Holidays"
140 DEF FN M(A)=((A/K2-INT (A/K2))*K2)
150 LET SV=0: LET MO=0: LET DA=0
160 PRINT "HAVE YOU ANY EXISTING
   LISTINGS Y/N ?": LET K$="yn"
170 CLS : GOSUB 990: CLS : LET P=2
180 IF C=1 THEN GOSUB 760
190 IF C=2 THEN GOSUB 1760
200 IF C=3 THEN GOSUB 2240
210 IF C>3 AND C<8 THEN LET
   KB=C-3: GOSUB 1140: LET SV=1
220 IF C=8 THEN GOSUB 1610: LET
   SV=0
230 IF C=9 AND SV=1 THEN PRINT :
   PRINT "YOU HAVE NOT SAVED ANY
   CHANGES""ARE YOU SURE YOU WANT
   TO QUIT": LET K$="yn": GOSUB 1480:
   IF KB=2 THEN LET C=0
240 IF C<>9 THEN GOTO 170
250 CLS : PRINT "GOODBYE"
260 STOP
270 LET MX=0: LET A2=0
280 LET K2=4: IF FN M(YR)=0 THEN LET
   A2=1
290 LET K2=100: IF FN M(YR)=0 THEN
   LET A2=0
300 LET K2=400: IF FN M(YR)=0 THEN
   LET A2=1
```

```
310 IF KB = 2 THEN LET MX = A2 + 28
320 IF KB = Ø THEN LET KB = 1
330 IF KB < > 2 THEN LET MX = D(KB)
340 LET KB = MX: RETURN
350 LET RS = Ø
360 IF DA = DE AND MO = ME AND KB = 5
    THEN LET KB = 5: RETURN
370 IF KB = 5 THEN LET KB = 7: RETURN
380 LET RS = Z(KB)
390 IF Q(KB) = Ø THEN GOTO 45Ø
4ØØ LET M4 = Q(KB)
41Ø FOR I = 1 TO M4
42Ø LET K$ = L$(KB,I): LET K2 = 3: GOSUB
    47Ø: IF VAL K$(2 TO 3) < > DA THEN LET
    K2 = Ø
43Ø IF K2 = 1 THEN LET KB = 7: RETURN
44Ø NEXT I
45Ø LET KB = RS
46Ø RETURN
47Ø IF KB < > 1 THEN GOTO 52Ø
48Ø IF VAL K$(1) = 3 THEN GOTO 54Ø
49Ø IF VAL K$(1) = 4 THEN GOTO 53Ø
5ØØ IF VAL K$(1) = 1 AND VAL K$(2 TO
    3) = DA THEN LET K2 = 1: RETURN
51Ø IF VAL K$(1) = 2 AND FN M((((YR −
    VAL K$(6 TO 9))*12) + (12 − VAL K$(4 TO
    5)) + MO)) = Ø THEN LET K2 = 1:
    RETURN
52Ø IF KB = 3 THEN GOTO 54Ø
53Ø IF VAL K$(2 TO 3) = DA AND VAL K$(4
    TO 5) = MO AND VAL K$(6 TO 9) = YR
    THEN LET K2 = 1: RETURN
535 LET K2 = Ø: RETURN
54Ø IF VAL K$(4 TO 5) = MO THEN LET
    K2 = 1: RETURN
55Ø LET K2 = Ø: RETURN
56Ø LET Y2 = Ø: LET D2 = Ø: LET M2 = Ø
57Ø LET Y2 = YR − 1
58Ø LET D2 = Y2*365 + INT (Y2/4) − INT
    (Y2/1ØØ) + INT (Y2/4ØØ)
59Ø IF MO = 1 THEN GOTO 63Ø
6ØØ FOR m = 1 TO MO − 1
61Ø LET KB = m: GOSUB 27Ø: LET
    D2 = D2 + KB
62Ø NEXT m
63Ø LET KB = D2 + DA: RETURN
64Ø LET MS = MO: LET DS = DA
65Ø LET DA = 1: LET MO = 3: GOSUB 56Ø:
    LET K2 = 7: LET DE = FN M(KB)
66Ø LET N2 = (INT (YR/1ØØ)) − 16: LET
    C2 = 3 + N2 − INT ((N2 + 1)/3) − INT
    (N2/4)
67Ø LET K2 = 19: LET N2 = FN M(YR + 1):
    LET K2 = 3Ø: LET D2 = FN
    M(C2 + (N2*19))
68Ø IF N2 > 11 AND D2 < 27 THEN LET
    D2 = D2 − 1: GOTO 7ØØ
69Ø IF N2 < = 11 AND D2 = 29 THEN LET
    D2 = 28
7ØØ LET D2 = D2 + 21
71Ø LET D2 = D2 + 1: LET K2 = 7: IF INT (FN
    M(D2 + DE) + Ø.1) < > 1 THEN
    GOTO 71Ø
72Ø IF D2 < 32 THEN LET ME = 3
73Ø IF D2 > = 32 THEN LET D2 = D2 − 31:
    LET ME = 4
74Ø LET DE = INT (D2 + Ø.1): LET MO = MS:
    LET DA = DS
75Ø RETURN
76Ø GOSUB 251Ø: GOSUB 248Ø
77Ø CLS
78Ø LET MK = 5
79Ø CLS
8ØØ PRINT AT 17,Ø;"<BREAK> returns to menu"
81Ø PRINT "z,x keys alter month.□"
82Ø PRINT INK 7;T$(1); INK 2;
    "□F□□"; INK 7;T$(2); INK 4;"□A"
83Ø PRINT INK 7;T$(3); INK 1;
    "□C□□"; INK 7;T$(4); INK 3;"□H"
84Ø PRINT AT Ø,Ø;
85Ø GOSUB 257Ø: IF MK < 5 THEN PRINT
    T$(MK)
86Ø PRINT #P: LET KB = 1: GOSUB 192Ø
87Ø IF P = 3 THEN PRINT #P
88Ø PRINT #P: LET T2 = MK: LET S2 = 1:
    GOSUB 2Ø2Ø
89Ø LET P = 2
9ØØ LET K$ = "zxfach□": GOSUB 148Ø: LET
    A = KB
91Ø IF A = 1 THEN LET MO = MO − 1
92Ø IF A = 2 THEN LET MO = MO + 1
93Ø IF MO = 13 THEN LET MO = 1: LET
    YR = YR + 1: GOSUB 64Ø
94Ø IF MO = Ø THEN LET MO = 12: LET
    YR = YR − 1: GOSUB 64Ø
95Ø IF A > 2 AND A < 7 THEN LET MK = A − 2
96Ø IF A < 3 THEN LET MK = 5
97Ø IF A < > 7 THEN GOTO 79Ø
98Ø RETURN
99Ø CLS : PRINT PAPER 5; INK 1; AT Ø,7;
    "□CALENDAR & DIARY□"; PAPER 6;
    INK Ø;AT 2,1Ø;"□MAIN MENU□"
1ØØØ FOR Z = 1 TO 19: PRINT PAPER 1;
    "□□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□
    □□□": NEXT Z: PRINT AT 3,Ø
1Ø1Ø PAPER 1: INK 7
1Ø2Ø PRINT AT 4,1;"1 . . . . . Look at monthly
    calendar"
1Ø3Ø PRINT AT 6,1;"2 . . . . . Look at yearly
    calendar"
```



```
1040 PRINT AT 8,1;"3.....Look at month
    diary"
1050 PRINT AT 10,1;"4.....Review/
    Edit Finance"
1060 PRINT AT 12,1;"5.....Review/
    Edit Appointments"
1070 PRINT AT 14,1;"6.....Review/
    Edit Celebrations"
1080 PRINT AT 16,1;"7.....Review/
    Edit Holidays"
1090 PRINT AT 18,1;"8.....Save the lists"
1100 PRINT AT 20,1; "9.....Leave the
    program"
1110 PRINT TAB 10;"please choose"
```




```
10 PRINT"■":DIM DL$(3,100):POKE
    53280,1:POKE 53281,1
20 M1$="JANUARY□□FEBRUARY□
    MARCH□□□□APRIL□□□□MAY
    □□□□□□JUNE□□□□□"
30 M2$="JULY□□□□□AUGUST
    □□□SEPTEMBEROCTOBER□□
    NOVEMBER□DECEMBER□"
40 MN$=M1$+M2$
50 ML$="312831303130313130313031"
60 DN$="SUNMONTUEWEDTHUFRISAT"
70 ST$(0)="FINANCIAL":ST$(1)=
    "APPOINTMENTS"
80 ST$(2)="CELEBRATIONS":ST$(3)=
    "HOLIDAYS"
90 CD$(1)="MNLY":CD$(2)="QRLY":
    CD$(3)="ANLY":CD$(4)="SNGL"
100 CL(0)=28:CL(1)=31:CL(2)=156:CL(3)
    =144:CL(4)=30:WH=5
110 Z=0:DR=0
120 GOSUB 2070
130 PP%=0
140 POKE 53280,6:POKE 53281,13:PRINT
    "■"TAB(9)"■■■□CALENDAR &
    PLANNER□■■"
150 PRINTTAB(10)"■■■□□□□□
    □□MENU□□□□□□□□■■"
160 PRINTTAB(12)"■■■■■■
    1 YEAR CALENDAR"
170 PRINTTAB(12)"■■2□SEE DIARY"
180 PRINTTAB(12)"■■3□ALTER DIARY"
190 PRINTTAB(12)"■■4□LEAVE
    PROGRAM"
200 PRINTTAB(9)"■■■■■■
    ■■■□□□□ENTER CHOICE
    □□□□■"
210 PRINTTAB(8)"■■■□□□□□
    □?□?□?□?□?□?□□□□□
    ■■"
220 GET A$:IF A$="" GOTO 220
230 A=VAL(A$)
240 IF A<1 OR A>4 GOTO 220
250 POKE 53280,1:ON A GOSUB
    290,290,1100,270
260 GOTO 130
270 IF DR=1 THEN GOSUB 2170
275 INPUT "■■PRESS (Q)UIT OR
    (M)ENU"; RS$:IF RS$="M" THEN
    RETURN
280 PRINT "■■":POKE 53280,
    14:POKE 53281,6:END
290 INPUT"■■■■ENTER YEAR (1753
    TO 29999)";Y
300 IF Y<1753 OR Y>29999 GOTO 290
310 YY=Y:Y=Y-1
320 LY=0
330 IF YY/4=INT(YY/4) THEN LY=1
340 IF YY/100=INT(YY/100) THEN LY=0
350 IF YY/400=INT(YY/400) THEN LY=1
360 ND=365*Y+INT(Y/4)-INT
    (Y/100)+INT(Y/400)+1
370 ED=ND+59+LY
380 ND=ND-INT(ND/7)*7
390 ED=ED-INT(ED/7)*7
400 N=INT(YY/100)-16
410 C=3+N-INT((N+1)/3)-INT(N/4)
420 N=YY+1-INT((YY+1)/19)*
    19:IF N=0 THEN N=19
430 D=C+19*N-INT((C+19*N)/30)*30
440 IF N>11 AND D>27 THEN D=D+1
450 IF N<=11 AND D=29 THEN D=28
460 D=D+21
470 D=D+1
480 X=D+ED-INT((D+ED)/7)*7
490 IF X<>1 THEN 470
500 EM=3:IF D>31 THEN
    D=D-31:EM=4
510 ED=D
520 IF A=1 THEN GOSUB 870:RETURN
530 INPUT"■■■■■WHICH
    MONTH";A$:M=VAL(A$):IF M<1 OR
    M>12 GOTO 530
540 GOSUB 550:RETURN
550 C=5:Z=0:PP=0
560 PRINT"■";CHR$(WH)
570 IF Z=0 THEN GOSUB 650
580 IF Z=1 THEN GOSUB 1870
590 PRINT"■■■■":GOSUB 2280:IF
    K=8 THEN K=0:GOTO 560
600 GOSUB 980
610 ON K GOSUB1600,1600,1600,
    1600,1830,1830,1850,2270
620 IF K=9 THEN RETURN
630 IF M>0 AND M<13 GOTO 560
640 RETURN
650 T=0
660 ML=VAL(MID$(ML$,2*M-1,2)):
    IF M=2 THEN ML=ML+LY
670 IF M=1 THEN T=0:GOTO 720
680 FOR N=1 TO M-1
690 T=T+VAL(MID$(ML$,2*N-1,2))
700 NEXT N
710 IF M>2 THEN T=T+LY
720 SD=T+ND-INT((T+ND)/7)*7
730 PRINT"■■■"TAB(11)
    "■"MID$(MN$,M*9-8,9);
    "□";YY;"■■":PRINT TAB(5);
```

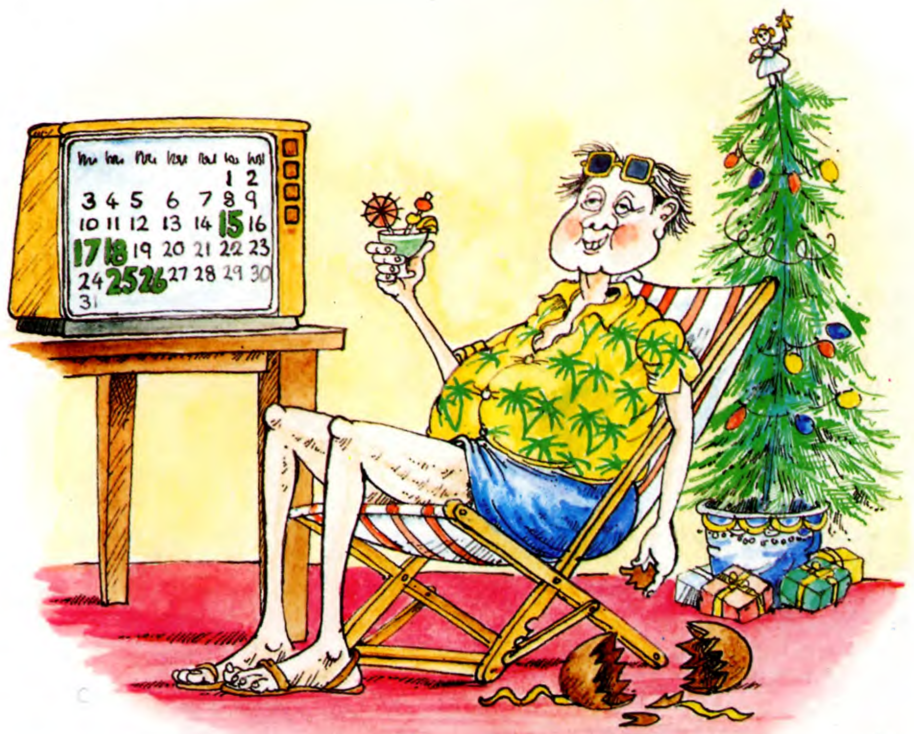
```
740 FOR N=1 TO 7:PRINT"■"
    MID$(DN$,3*N-2,3);"■□■";:
    NEXT N
750 PRINT"■":PRINTTAB
    (4*SD+5);
760 X=SD
770 FOR CD=1 TO ML
780 GOSUB 1600
790 D$=STR$(CD)+"□"
800 IF LEN(D$)<4 THEN D$=D$+
    CHR$(32)
810 PRINTCHR$(CC);D$;:X=X+1
820 IF X>6 THEN PRINT:
    PRINTTAB(5);:X=0
830 NEXT CD
840 IF M=EM THEN PRINT:
    PRINTTAB(7)"■■EASTER
    SUNDAY□";ED;"□";MID$(MN$,
    M*9-8,9)
850 PRINT
860 RETURN
870 C=5
880 PRINT"■■■■■□□□□□□DO
    YOU WANT A PRINTOUT
    (Y/N)?□□□□"
890 GET A$:IF A$="" GOTO 890
900 IF A$="Y" THEN GOSUB 2270
910 FOR M=1 TO 12
920 GOSUB 650
930 PRINT
940 IF PP=0 THEN GETA$:IF A$="" GOTO
    940
950 NEXT M
960 GOSUB 2280
970 RETURN
980 PRINT"■■■■■■■F■■
    INANCE,■■A■■PPOINTMENTS,
    ■■H■■OLIDAYS"
990 PRINTTAB(7)"■■C■■
    ELEBRATIONS,■N■EXT,
    ■L■AST"
1000 PRINTTAB(11)"■S■WAP,
    ■P■RINT■M■ENU"
1010 GET A$:IF A$="" GOTO 1010
1020 N=0
1030 N=N+1
1040 IF MID$("FACHNLSPM",N,1)
    =A$ GOTO 1070
1050 IF N<9 GOTO 1030
1060 C=0:GOTO1010
1070 K=N:C=N-1:IFC<4 THEN CP=C
1080 IF Z=1 AND K<5 GOTO 1010
1090 RETURN
```



```
10 MODE7
20 *FX229,1
22 *OPT2,1
24 *OPT1,1
30 *FX4,1
40 DIMList$(3,150),Type$(3), code%(4)
50 DayMonth$="3128313031303131
    30313031"
60 MonthName$="January□□
    February□March□□□□April
    □□□□May□□□□□□June
    □□□□□July□□□□□
    August□□□SeptemberOctober
    □□November□December□"
70 DayName$="SunMonTueWedThu
    FriSat"
80 Pay$="MnlyQrlyAnlySngl"
90 Type$(0)="Finances":Type$(1)
    ="Appointments":Type$(2)
    ="Celebrations":Type$(3)
    ="Holidays"
100 F$=CHR$132+CHR$157+CHR$
    135+CHR$141
110 SaveFlag%=0:P%=0:Month%
    =0:Day%=0
120 PRINT"Have you existing lists (Y/N)?"
130 REPEAT
140 MODE7:C%=FNmenu:MODE7
150 IF C%=1 PROCscrM
160 IF C%=2 PROCannual
170 IF C%=3 PROCdiary
180 IF C%>3 AND C%<8
    PROClist(C%-4):SaveFlag%=1
190 IF C%=8 PROCsave:SaveFlag% =0
200 IF C%=9 AND SaveFlag%=1
    PRINT''"You have not saved any
    alterations":PRINT"Are you sure you wish to
    leave (Y/N) ":IF FNget("YN")=2 C%=0
210 UNTIL C%=9
220 *FX4,0
230 *FX229
240 CLS:PRINTTAB(12,12)"GOODBYE"
250 END
260 DEF FNmonthL(m%)
270 LOCAL max%,a%
280 IF Year%MOD4=0 a%=1
290 IF Year%MOD100=0 a%=0
300 IF Year%MOD400=0 a%=1
310 IF m%=2 max%=a%+28
320 IF m%<>2 max%=VAL(MID$
    (DayMonth$,m%*2-1,2))
330 =max%
340 DEF FNmarker(type%)
350 LOCALa$,n%,flag%,max%
360 IF P%=2 =32
370 IF type%=5 AND Month%=Meast%
    AND Day%=Deast% =134
380 IF type%=5 =135
390 max%=VAL(List$(type%,0))
400 REPEAT
410 n%=n%+1
420 a$=List$(type%,n%)
```

```
430 d%=ASC(MID$(a$,2,1))
440 IF d%<>Day% GOTO46Ø
450 flag%=FNcheck(a$)
460 UNTIL flag%=1 OR n%>=max%
470 IF flag%=1 =135
480 n%=129+type%:IF n%=132 =133
490 IF n%=131 =132
500 =n%
510 DEF FNcheck(a$)
520 LOCALt%,y%,m%,flag%
530 t%=ASC(MID$(a$,1,1))
540 y%=ASC(MID$(a$,3,1))
550 m%=y%AND&F
560 y%=(y%DIV16+17)*1ØØ+ASC
    (MID$(a$,4,1))
570 IF y%>Year% =Ø
580 IF (t%=1 AND Month%>=m%) OR
    (t%=2 AND (m%-Month%)MOD3=Ø)
    OR (t%=3 AND m%=Month%) OR
    (t%=4 AND m%=Month% AND
    y%=Year%) flag%=1
590 =flag%
600 DEF FNdayNo
610 LOCAL d%,m%,y%
620 y%=Year%-1
630 d%=y%*365+y%DIV4-y%DIV
    1ØØ+y%DIV4ØØ
640 IF Month%=1 GOTO 68Ø
650 FOR m%=1 TO Month%-1
660 d%=d%+FNmonthL(m%)
670 NEXT
680 =d%+Day%
690 DEF PROCeaster
700 LOCALc%,n%,d%
710 Mstore%=Month%:Dstore%=Day%
720 Day%=1:Month%=3: Deast%
    =FNdayNo MOD7
730 n%=(Year%DIV1ØØ)-16:c%
    =3+n%-(n%+1)DIV3
    -n%DIV4
740 n%=(Year%+1)MOD19:d%=
    (c%+(n%*19))MOD3Ø
750 IF n%>11 AND d%>27 d%=
    d%-1 ELSE IF n%<=11 AND d%=29
    d%=28
760 d%=d%+21:REPEAT d%=
    d%+1:UNTIL (d%+Deast%) MOD7=1
770 IF d%<32 Meast%=3 ELSE
    d%=d%-31:Meast%=4
780 Deast%=d%:Month%=Mstore%:
    Day%=Dstore%
790 ENDPROC
800 DEF PROCscrM
810 LOCAL a,a$,t
820 PROCmydate:PROCprinter
830 FOR t=19 TO 24
840 PRINTTAB(Ø,t)CHR$132+CHR$
    157+CHR$135;:PRINTTAB(36,t)
    CHR$156;
850 NEXT
860 VDU28,3,24,34,19
870 PRINTTAB(Ø,Ø)"Use Cursor arrows to alter
    month"
880PRINT"<ESCAPE> to return to Menu"
890 PRINT'Type$(Ø);"□□";Type$(1)
900 PRINTType$(2);"□□";Type$(3);
910 VDU26,28,Ø,18,39,Ø
920 marker%=5
930 REPEAT
940 CLS:*FX15
950 PROCmyheader:PRINT:
    PROCprintdays(1)
960 PRINT':PROCprintmonth(marker%,1)
970 P%=Ø:VDU3
980 a=FNget(CHR$136+CHR$137+
    "FACH"+CHR$27)
990 IF a=1 Month%=Month%-1
1000 IF a=2 Month%=Month%+1
1010 IF Month%=13:Month%=1:Year
    %=Year%+1:PROCeaster
1020 IF Month%=Ø:Month%=12:
    Year%=Year%-1:PROCeaster
1030 IF a>2 AND a<7 marker%=a-3
1040 IF a<3 marker%=5
1050 UNTIL a=7
1060 ENDPROC
1070 DEF FNmenu
1080 LOCAL n
1090 FOR n=1 TO 2
1100 PRINTF$+"INPUT :□"+CHR$
    135+"Calendar & Diary□□□
    □□"+CHR$156
1110 NEXT
1120 PRINT
1130 FOR n=1 TO 2
1140 PRINTSPC(9)F$+"Menu□□□"
    +CHR$156
1150 NEXT
1160 PRINT'"1"+CHR$131+"Look at
    Monthly Calendar"
1170 PRINT"2"+CHR$131+"Look at Year
    Calendar"
1180 PRINT"3"+CHR$131+"Look at Month
    Diary"
1190 PRINT"4"+CHR$131+"Review/
    Edit Finance"
1200 PRINT"5"+CHR$131+"Review/
    Edit Appointments"
1210 PRINT"6"+CHR$131+"Review/
    Edit Celebrations"
1220 PRINT"7"+CHR$131+"Review/
    Edit Holidays"
1230 PRINT"8"+CHR$131+
    "Save the lists"
1240 PRINT"9"+CHR$131+"Leave the
    program"
1250 PRINTTAB(18,18)"Please Choose□";
1260 =FNget("123456789")
```

```
10 CLS
20 CLEAR 5ØØØ
30 DIM LI$(3,15Ø),TY$(3),CO(4)
40 DM$="312831303130313l
   3Ø313Ø31"
50 MN$="JANUARY□□FEBRUARY□
   MARCH□□□□APRIL□□□□MAY
   □□□□□□JUNE□□□□□JULY
   □□□□□AUGUST□□□
```



```
   SEPTEMBEROCTOBER□□NOVEMBER□
   DECEMBER□"
6Ø DN$="SUNMONTUEWEDTHUFRISAT"
7Ø PA$="MNLYQRLYANLYSNGL"
8Ø TY$(Ø)="FINANCES":TY$(1)=
   "APPOINTMENTS":TY$(2)=
   "CELEBRATIONS":TY$(3)="HOLIDAYS"
9Ø DEF FNM(A)=INT((A/K2-INT(A/K2))*
   K2+Ø.5)*SGN(A/K2)
1ØØ SV=Ø:P=Ø:MO=Ø:DA=Ø
11Ø PRINT@256,"HAVE YOU EXISTING LISTS
   (Y/N)?"
12Ø REM
13Ø CLS:GOSUB 1Ø3Ø:CLS:P=Ø
14Ø IF C=1 GOSUB 77Ø
15Ø IF C=2 GOSUB 2Ø1Ø
16Ø IF C=3 GOSUB 246Ø
17Ø IF C>3 AND C<8 THEN KB=C-4:
   GOSUB 118Ø:SV=1
18Ø IF C=8 GOSUB 173Ø:SV=Ø
19Ø IF C=9 AND SV=1 THEN PRINT:
   PRINT"YOU HAVE NOT SAVED ANY
   CHANGES": PRINT"ARE YOU SURE YOU
   WANT TO QUIT": KB$="YN": GOSUB
   159Ø:IF KB=2 THEN C=Ø
2ØØ IF C<>9 THEN 12Ø
21Ø CLS:PRINT"GOOD-BYE"
22Ø END
23Ø 'GET LENGTH OF MONTH
24Ø MX=Ø:A2=Ø
25Ø K2=4:IF FNM(YR)=Ø THEN A2=1
26Ø K2=1ØØ:IF FNM(YR)=Ø THEN A2=Ø
27Ø K2=4ØØ:IF FNM(YR)=Ø THEN A2=1
28Ø IF KB=2 THEN MX=A2+28
29Ø IF KB<>2 THEN MX=VAL(MID$
   (DM$,KB*2-1,2))
3ØØ KB=MX:RETURN
31Ø 'GET MARKER CHARACTER
32Ø A3$="":N3=Ø:F3=Ø:M3=Ø
33Ø IF P=2 THEN RS=32:GOTO 46Ø
34Ø IF KB=5 AND MO=ME AND DA=DE
   THEN RS=191:GOTO 46Ø
35Ø IF KB=5 THEN RS=143:GOTO46Ø
36Ø M3=VAL(LI$(KB,Ø))
37Ø REM
38Ø N3=N3+1
39Ø A3$=LI$(KB,N3)
4ØØ IF MID$(A3$,2,1)="" THEN D=Ø ELSE
   D=ASC(MID$(A3$,2,1))
41Ø IF D<>DA THEN 43Ø
42Ø KB$=A3$:GOSUB 47Ø:F3=K2
43Ø IF NOT(F3=1 OR N3>M3) THEN 37Ø
44Ø IF F3=Ø THEN RS=32:GOTO 46Ø
45Ø N3=159+16*KB:RS=N3
46Ø KB=RS:RETURN
47Ø 'CHECK FOR ITEM IN MONTH
48Ø T4=Ø:Y4=Ø:F4=Ø
49Ø T4=ASC(MID$(KB$,1,1))
5ØØ Y4=ASC(MID$(KB$,3,1))
51Ø M4=(Y4 AND 15)
52Ø Y4=(FIX(Y4/16)+17)*1ØØ+
```

Is it easy to change the titles of the categories for the diary entries or add extra ones?

Changing the names is quite easy and allows you to tailor the diary to suit your specific needs. You'll have to alter every occurence of the words and their initial letters in the program.

For the Spectrum change Lines 13Ø, 82Ø, 83Ø, 9ØØ and 1Ø5Ø to 1Ø8Ø; for the Commodore, Lines 7Ø, 8Ø, 98Ø and 1Ø4Ø; for the Acorn, Lines 9Ø, 98Ø and 119Ø to 122Ø; and for the Dragon, Lines 8Ø, 94Ø and 11ØØ to 113Ø.

Remember, though, that the program is designed to allow regular entries under the Finance and Celebrations categories—whatever new names you call them. So arrange your new diary with this in mind.

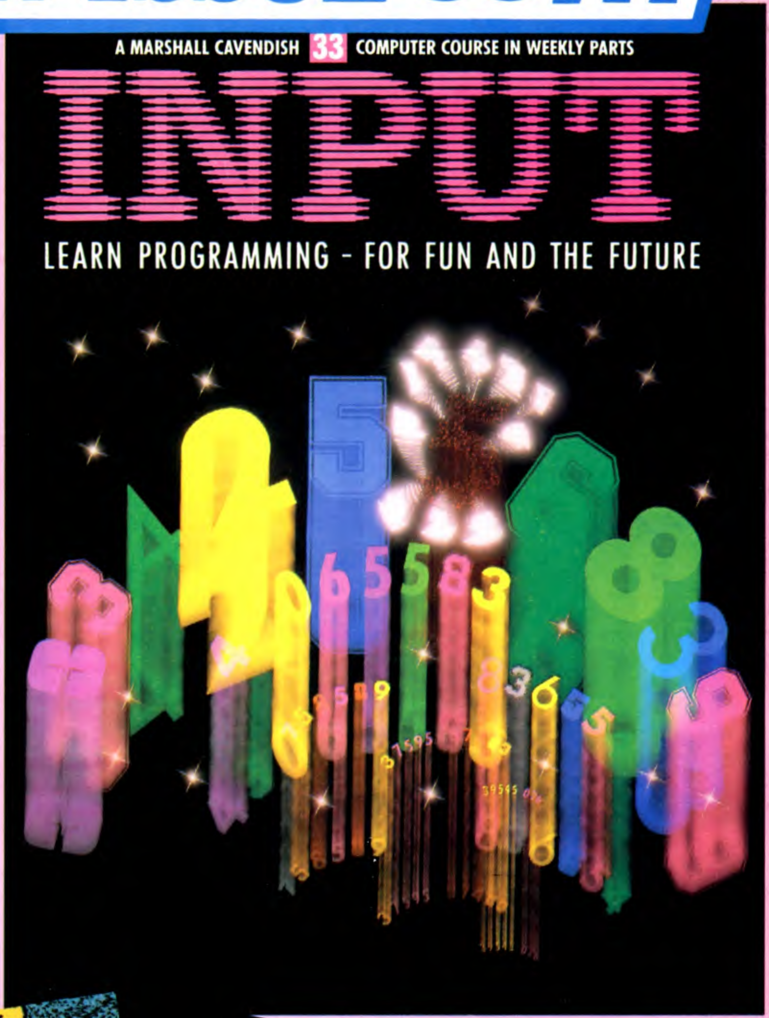
Adding extra categories is more difficult and would mean altering many routines. It is not worth attempting unless you are an experienced programmer.

```
   ASC(MID$(KB$,4,1))
53Ø IF Y4>YR THEN K2=Ø:RETURN
54Ø K2=3:IF (T4=1 AND MO>=M4) OR
   (T4=2 AND FNM(M4-MO)=Ø) OR
   (T4=3 AND M4=MO) OR (T4=4 AND
   M4=MO AND Y4=YR) THEN F4=1
55Ø K2=F4:RETURN
56Ø 'GET DAY NO.
57Ø YX=Ø:D2=Ø:M2=Ø
58Ø Y2=YR-1
59Ø D2=Y2*365+FIX(Y2/4)-FIX
   (Y2/1ØØ)+FIX(Y2/4ØØ)
6ØØ IF MO=1 THEN 64Ø
61Ø FOR M2=1 TO MO-1
62Ø KB=M2:GOSUB 23Ø:D2=D2+KB
63Ø NEXT
64Ø KB=D2+DA:RETURN
65Ø 'FIND EASTER DATE
66Ø N2=Ø:C2=Ø:D2=Ø
67Ø MS=MO:DS=DA
68Ø DA=1:MO=3:GOSUB56Ø:K2=7:
   DE=FNM(KB)
69Ø N2=FIX(YR/1ØØ)-16:C2=3+
   N2-FIX((N2+1)/3)-FIX(N2/4)
7ØØ K2=19:N2=FNM(YR+1):K2=3Ø:
   D2=FNM(C2+(N2*19))
71Ø IF N2>11 AND D2<27 THEN
   D2=D2-1 ELSE IF N2<=11 AND
   D2=29 THEN D2=28
72Ø D2=D2+21
73Ø D2=D2+1:K2=7:IF
   FNM(D2+DE)<>1THEN 73Ø
74Ø IF D2<32 THEN ME=3 ELSE
   D2=D2-31:ME=4
75Ø DE=D2:MO=MS:DA=DS
76Ø RETURN
77Ø 'LOOK AT MON CAL
78Ø REM
79Ø GOSUB 275Ø:GOSUB 272Ø
8ØØ CLS
81Ø PRINT,"UP/DOWN ARROWS ALTER
   MONTH"
82Ø PRINT"USE clear TO RETURN TO MENU"
83Ø PRINT:PRINT CHR$(159);TY$(Ø),
   CHR$(175);TY$(1)
84Ø PRINT:PRINT CHR$(191);TY$(2),
   CHR$(2Ø7);TY$(3)
85Ø PRINT:PRINT"PRESS ANY KEY TO
   CONTINUE..."
855 IF INKEY$=""THEN 855
86Ø MK=5
87Ø REM
88Ø CLS
89Ø GOSUB 282Ø:IF MK<4 THEN
   PRINT@19,TY$(MK)
9ØØ PRINT#-P:KB=1:GOSUB 215Ø
91Ø IF P=2 THEN PRINT#-P
92Ø PRINT#-P:T2=MK:S2=1:GOSUB 224Ø
93Ø P=Ø
94Ø KB$="↑"+CHR$(1Ø)+"FACH"+
   CHR$(12):GOSUB 159Ø:A=KB
95Ø IF A=1 THEN MO=MO-1
96Ø IF A=2 THEN MO=MO+1
97Ø IF MO=13 THEN MO=1:YR=YR+1:
   GOSUB 65Ø
98Ø IF MO=Ø THEN MO=12:YR=YR-1:
   GOSUB 65Ø
99Ø IFA>2 AND A<7 THEN MK=A-3
1ØØØ IF A<3 THEN MK=5
1Ø1Ø IF A<>7 THEN 87Ø
1Ø2Ø RETURN
1Ø3Ø 'MENU RETURN CHOICE IN KB
1Ø4Ø PRINT"□□□□CALENDAR & DIARY
   PROGRAM□□□□□□□□□";
   STRING$(24,131)
1Ø5Ø PRINT TAB(13);"menu"
1Ø6Ø PRINT
1Ø7Ø PRINT"1: LOOK AT MONTHLY
   CALENDAR"
1Ø8Ø PRINT"2: LOOK AT YEAR CALENDAR"
1Ø9Ø PRINT"3: LOOK AT MONTH DIARY"
11ØØ PRINT"4: REVIEW/EDIT FINANCE"
111Ø PRINT"5: REVIEW/EDIT
   APPOINTMENTS"
112Ø PRINT"6: REVIEW/EDIT CELEBRATIONS"
113Ø PRINT"7: REVIEW/EDIT HOLIDAYS"
114Ø PRINT"8: SAVE THE LISTS"
115Ø PRINT"9: LEAVE THE PROGRAM"
116Ø PRINT:PRINTTAB(9);"PLEASE CHOOSE"
117Ø KB$="123456789":GOSUB 159Ø:
   C=KB:RETURN
```

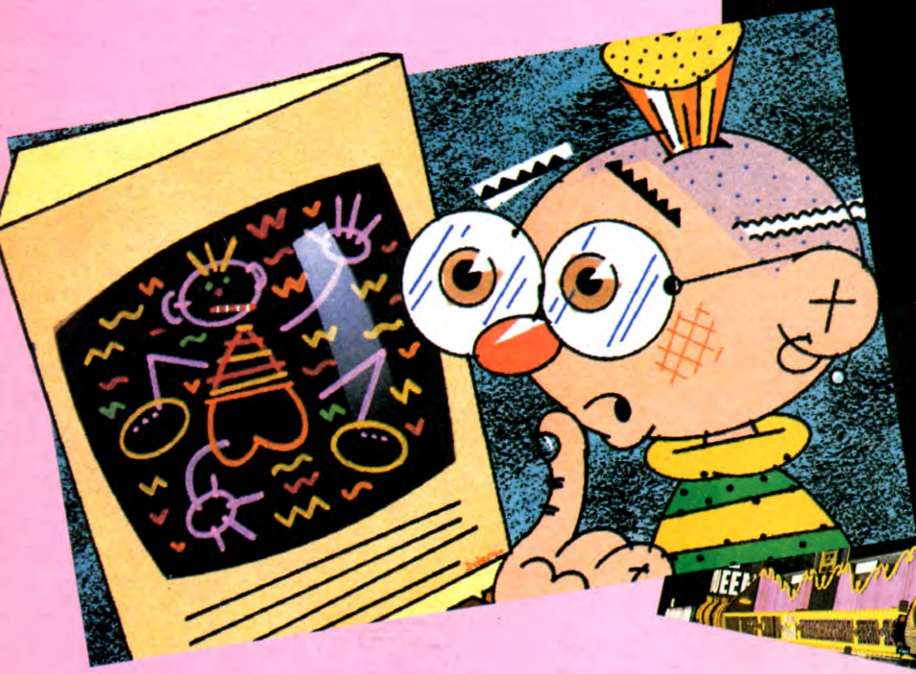
CUMULATIVE INDEX

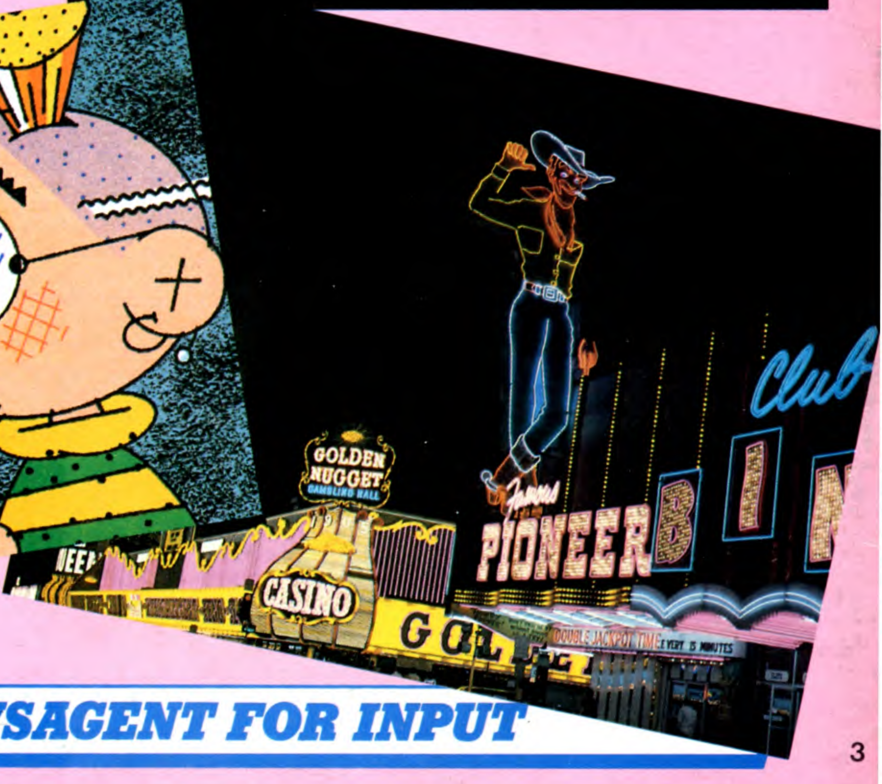
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 33...



- *Learn how to use **PAGED GRAPHICS**. Related to cartoon films and 'flicker books', it is a really useful technique to add to your animation armoury*
- *Delve further into the secret world of codes. Learn about **CODE BREAKING** and how to use more sophisticated **CODING TECHNIQUES***
- *More and more electronic one-armed bandits are infiltrating the arcades. Start entering the **FRUIT MACHINE** program. If you are addicted to push-button gambling, this program could save you a fortune!*
- *The stage is set for **CLIFFHANGER**. Now is the time to push the title page aside and bring on the cliff and the sky by adding the **SCROLL ROUTINES***
- *Is your life a seething morass of confusion? Don't worry, help is at hand with part two of the **CALENDAR** and **DIARY** program*





ASK YOUR NEWSAGENT FOR INPUT